GaussDB
2.23.06.000

Centralized-Developer Guide

Issue 01

Date 2023-06-13

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Huawei Cloud Computing Technologies Co., Ltd.

Address:	Huawei Cloud Data Center Jiaoxinggong Road Qianzhong Avenue Gui'an New District Gui Zhou 550029 People's Republic of China
Website:	https://www.huaweicloud.com/intl/en-us/

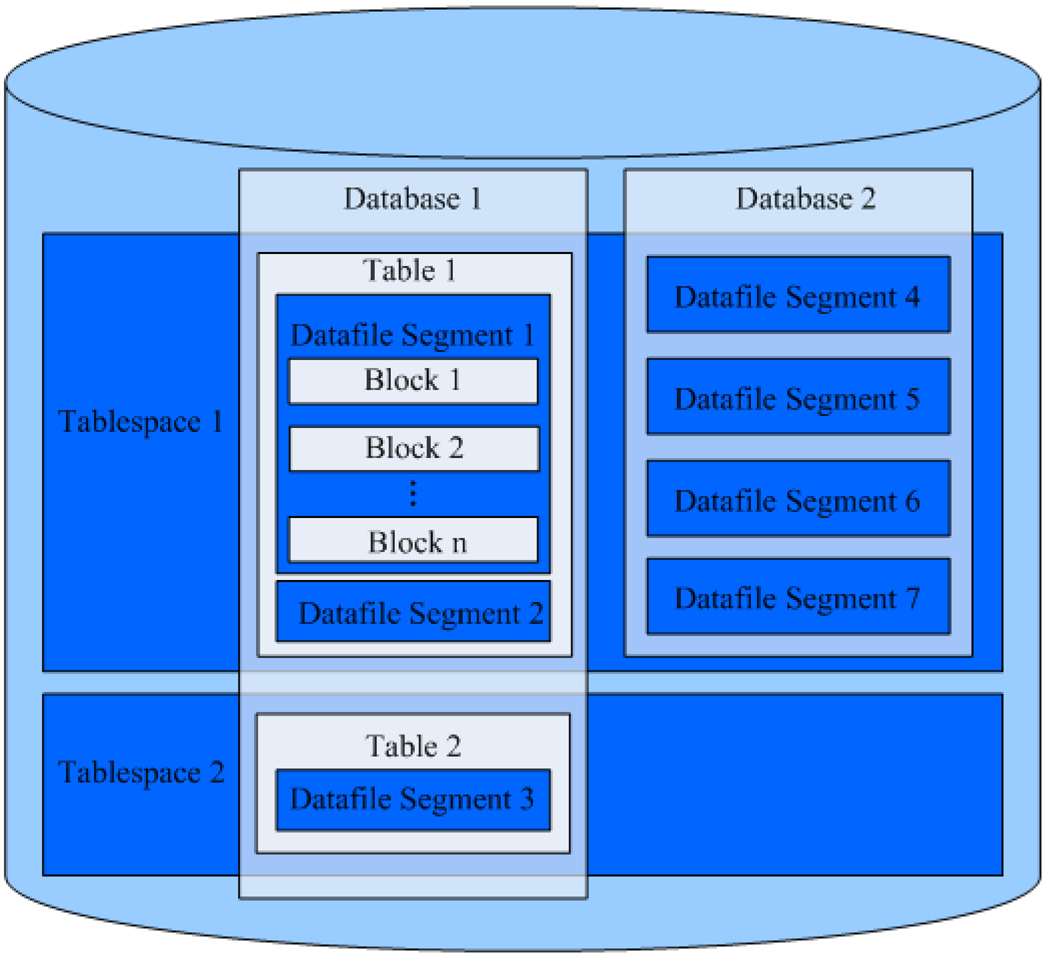
Contents

1 Overview

1.1 Database Logical Architecture

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

Figure 1-1 Database logical architecture

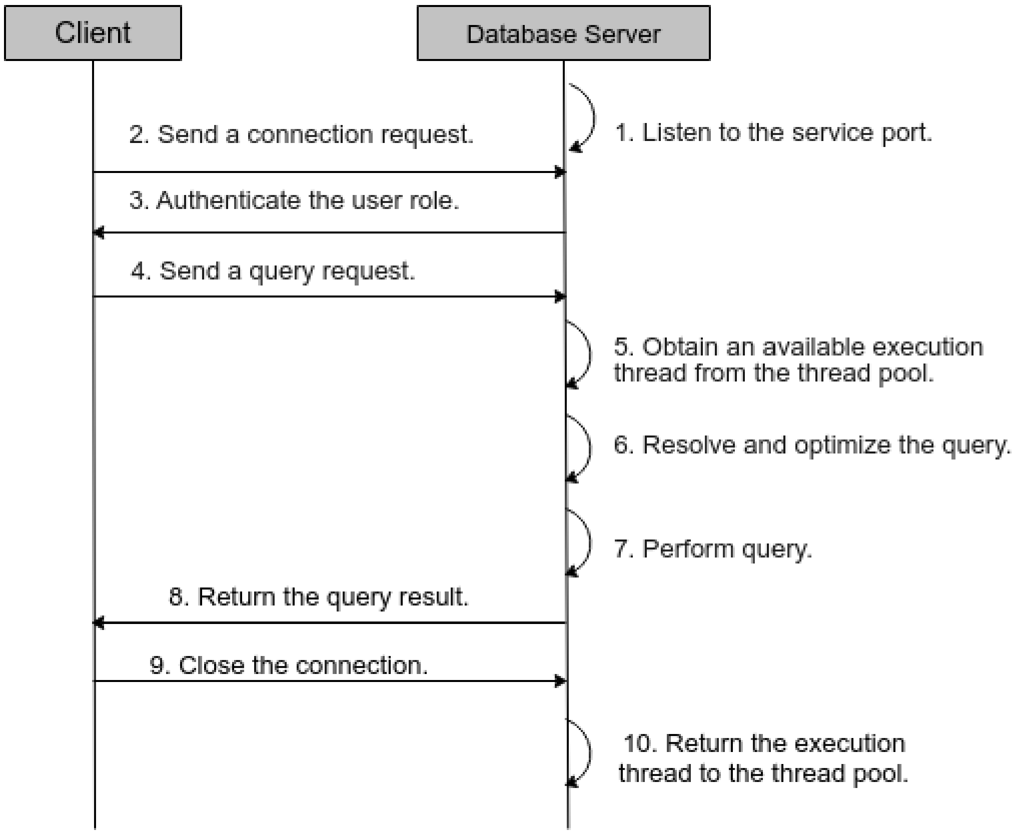


NOTE

- Tablespace: Directory storing physical files of its databases. Multiple tablespaces can coexist, and each of them can contain files belonging to different databases.
- Database: A database manages various data objects and is isolated from each other. Objects managed by a database can be distributed to multiple tablespaces.
- Datafile Segment: Data file, each of which stores data of only one table. A table containing more than 1 GB of data is stored in multiple datafile segments.
- One table belongs to only one database and one tablespace. The datafile segments storing the data of the same table must be in the same tablespace.
- Block: Basic unit of database management. Its default size is 8 KB.

1.2 Query Request Handling Process

Figure 1-2 GaussDB service response process


Client
Database Server
1. Listen to the service port.
2. Send a connection request.
3. Authenticate the user role.
4. Send a query request.
5. Obtain an available execution thread from the thread pool.
6. Resolve and optimize the query.
7. Perform query.
8. Return the query result.
9. Close the connection.
10. Return the execution thread to the thread pool.

1.3 Managing Transactions

A transaction is a customized sequence of database operations, which form an integral unit of work. In GaussDB, you can start, set, commit, and roll back transactions. A GaussDB database supports the following transaction isolation levels: READ COMMITTED, READ UNCOMMITTED (not recommended), REPEATABLE READ, and SERIALIZABLE. SERIALIZABLE is equivalent to REPEATABLE READ.

Controlling Transactions

The following describes transaction operations supported by the database:

- Starting transactions

  You can start transactions using **START TRANSACTION** and **BEGIN**.

- Setting transactions

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

- Committing transactions

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

- Rolling back transactions

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

Transaction Isolation Levels

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

NOTE

The isolation level cannot be changed after data is modified using **SELECT**, **INSERT**, **DELETE**, **UPDATE**, **FETCH**, or **COPY** in the transaction.

- **READ COMMITTED**: At this level, a transaction can access only committed data. This is the default level.

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

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

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

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

1.4 Concepts

Database

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

Tablespace

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

Schema

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

User and Role

GaussDB uses users and roles to control the access to databases. A role can be a database user or a group of database users, depending on role settings. In GaussDB, the difference between roles and users is that a role does not have the **LOGIN** permission by default. In GaussDB, one user can have only one role, but you can put a user's role under a parent role to grant multiple permissions to the user.

Transaction

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

2 Working with Databases

2.1 Connecting to a Database

Client tools for connecting to a database include DAS, gsql, and APIs (such as ODBC and JDBC).

- DAS enables you to manage databases on a web-based console. It supports SQL execution, advanced database management, and intelligent O&M, simplifying database management and improving efficiency and data security. The permissions required for connecting to instances through DAS are enabled by default. You are advised to use DAS to connect to instances. For details about how to connect to GaussDB, see **Data Admin Service User Guide**.
- gsql is a client tool provided by GaussDB. You can use gsql to connect to the database and then enter, edit, and execute SQL statements in an interactive manner. For details, see **Connecting to a DB Instance**.
- You can use standard database APIs in **APIs**, such as ODBC and JDBC, to develop GaussDB-based applications.

NOTICE

- In the primary/standby deployment, you can use a client tool to connect to a database through a primary DN. Before connection, you must obtain the IP address and port number of the server where the primary DN is deployed. You are not advised to connect a database through a standby DN when services are running properly.

2.1.1 APIs

You can use standard database APIs, such as **ODBC** and **JDBC**, to develop GaussDB-based applications.

Supported APIs

Each application is an independent GaussDB development project. APIs alleviate applications from directly operating in databases, and enhance the database portability, extensibility, and maintainability. **Table 2-1** lists the APIs supported by GaussDB and the download addresses.

Table 2-1 Database APIs

API	How to Obtain
ODBC	● Linux: Driver: GaussDB-Kernel_VxxxRxxxCxx.x-xxxxx-64bit-Odbc.tar.gz unixODBC source code package: **http://sourceforge.net/projects/unixodbc/files/unixODBC/2.3.0/unixODBC-2.3.0.tar.gz/download** ● Windows: Driver: GaussDB-Kernel_VxxxRxxxCxx.x-Windows-Odbc.tar.gz
JDBC	● Driver: GaussDB-Kernel_VxxxRxxxCxx.x-xxxxx-64bit-Jdbc.tar.gz ● Driver class: org.postgresql.Driver

For details about more APIs, see **Application Development Guide**.

2.2 Before You Start

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

Prerequisites

GaussDB is running properly.

Procedure

Step 1 Connect to a database. For details, see **Connecting to a Database**.

Step 2 Create a database user.

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

```
openGauss=# CREATE USER joe WITH PASSWORD "xxxxxxxxx";
```

If the following information is displayed, the table has been created:

```
CREATE ROLE
```

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

Run the following command to set user **joe** as the system administrator.

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

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

Note: For details about how to create users, see **Managing Users and Their Permissions**.

Step 3 Create a database.

```
openGauss=# CREATE DATABASE db_tpcc OWNER joe;
```

If the following information is displayed, the database has been created:

```
CREATE DATABASE
```

After **db_tpcc** database is created, you can run the following command to exit the **postgres** database and log in to the **db_tpcc** database as the user you created for more operations. You can also continue using the default **postgres** database.

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

db_tpcc=>
```

Create a schema.

```
db_tpcc=> CREATE SCHEMA joe AUTHORIZATION joe;
```

If the following information is displayed, the schema has been created:

```
CREATE SCHEMA
```

Note:

New databases are created in the **pg_default** tablespace by default. To specify another tablespace, run the following statement:

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

hr_local indicates the tablespace name. For details about how to create a tablespace, see **Creating and Managing Tablespaces**.

Step 4 Create a table.

- Create a table named **mytable** that has only one column. The column name is **firstcol** and the column type is **integer**.
```
db_tpcc=>  CREATE TABLE mytable (firstcol int);
CREATE TABLE
```
- Insert data into the table.
```
db_tpcc=> INSERT INTO mytable values (100);
```
  If the following information is displayed, the data has been inserted:
```
INSERT 0 1
```
- View the data in the table.
```
db_tpcc=> SELECT * from mytable;
 firstcol
----------
      100
(1 row)
```

Note:

- By default, new database objects, such as the newly created table, are created in the **$user** schema. For more details about schemas, see **Creating and Managing Schemas**.
- For more details about how to create a table, see **Creating and Managing Tables**.

- In addition to the created tables, a database contains many system catalogs. These system catalogs contain database installation information and information about various queries and processes in GaussDB. You can collect information about the database by querying system catalogs. For details, see **Querying a System Catalog**.

  GaussDB supports hybrid row-column store, providing high query performance for interaction analysis in complex scenarios. For details about how to select a store method, see **Planning a Storage Model**.

----End

2.3 Creating and Managing Databases

Prerequisites

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

Background

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

  NOTE

  - The template database does not contain any user table. You can view the attributes of the template database in the **PG_DATABASE** system catalog.
  - The **template0** template does not allow user connections. Only the initial user of the database and the system administrator can connect to **template1**.
- A maximum of 128 databases can be created in GaussDB.
- A database system consists of multiple databases. A client can connect to only one database at a time. Users cannot query data across databases. If one GaussDB contains multiple databases, set the **-d** parameter to specify the database instance to be connected.

Precautions

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

To resolve this problem, you need to:

- Ensure that the name of the data object does not exceed the maximum length.
- Change the default database storage code set (**server_encoding**) to UTF-8.
- Exclude multi-byte characters from object names.
- Ensure that no more than 128 databases are created (recommended).
- If you fail to delete an object by specifying its name after truncation, specify its original name to delete it, or manually delete it from the system catalogs on each node.

Procedure

Step 1 Run the following command to create a database named **db_tpcc**:

```
openGauss=# CREATE DATABASE db_tpcc;
CREATE DATABASE
```

NOTE

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

Step 2 View databases.

- Run the **\l** meta-command to view the database list of the database system.
  ```
  openGauss=# \l
  ```
- Run the following command to query the database list in the **pg_database** system catalog:
  ```
  openGauss=# SELECT datname FROM pg_database;
  ```

Step 3 Modify the database.

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

- Run the following command to set the default search path for the database:
  ```
  openGauss=# ALTER DATABASE db_tpcc SET search_path TO pa_catalog,public;
  ALTER DATABASE
  ```
- Run the following command to rename the database:
  ```
  openGauss=# ALTER DATABASE db_tpcc RENAME TO human_tpcds;
  ALTER DATABASE
  ```

Step 4 Delete the database.

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

Run the following command to delete the database:

```
openGauss=# DROP DATABASE human_tpcds;
DROP DATABASE
```

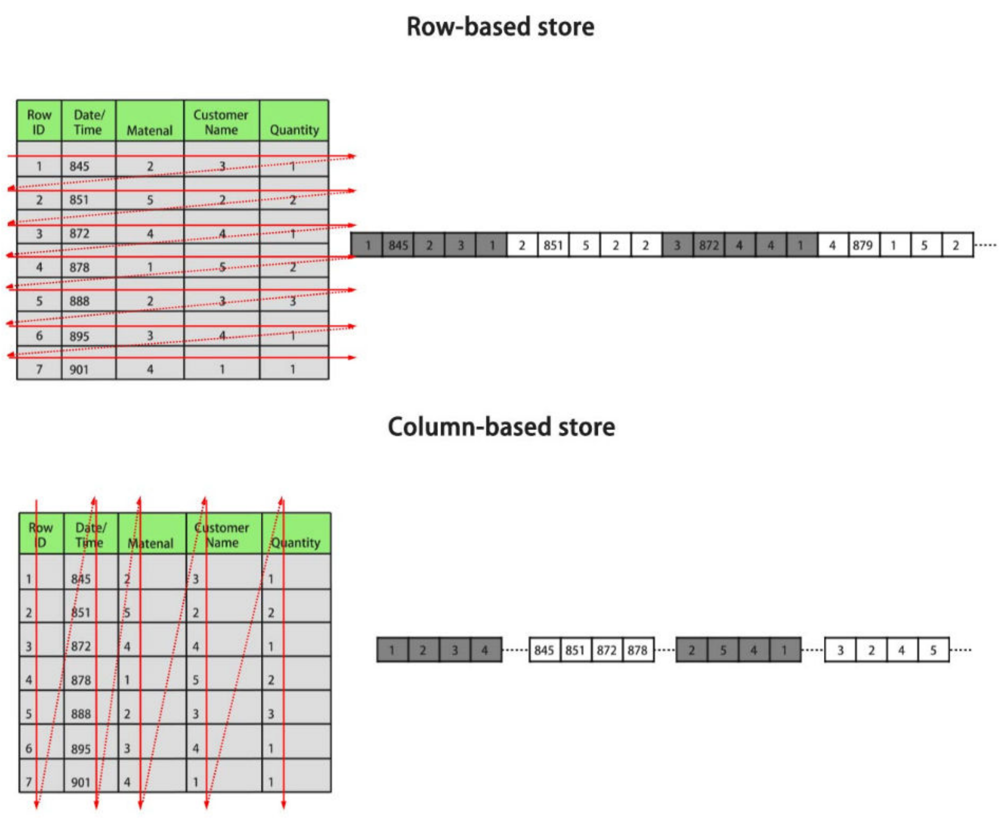
----End

2.4 Planning a Storage Model

GaussDB supports hybrid row-column storage. Each storage model applies to specific scenarios. Select an appropriate model when creating a table. Generally, GaussDB is used for transactional processing databases. By default, row storage is used. Column storage is used only when complex queries in large data volume are performed.

Row-store stores tables to disk partitions by row, and column-store stores tables to disk partitions by column. By default, a row-store table is created. For details about differences between row storage and column storage, see **Figure 2-1**.

Figure 2-1 Differences between row storage and column storage



Row ID	Date/ Time	Matenal	Customer Name	Quantity
1	845	2	3	1
2	851	5	2	2
3	872	4	4	1
4	878	1	5	2
5	888	2	3	3
6	895	3	4	1
7	901	4	1	1

Row ID	Date/ Time	Matenal	Customer Name	Quantity
1	845	2	3	1
2	851	5	2	2
3	872	4	4	1
4	878	1	5	2
5	888	2	3	3
6	895	3	4	1
7	901	4	1	1

In the preceding figure, the upper left part is a row-store table, and the upper right part shows how the row-store table is stored on a disk; the lower left part is a column-store table, and the lower right part shows how the column-store table is stored on a disk.

Both storage models have benefits and drawbacks.

Storage Model	Benefit	Drawback
Row storage	Record data is stored together. Data can be easily inserted and updated.	All the columns of a record are read after the **SELECT** statement is executed even if only certain columns are required.
Column storage	● Only the columns involved in a query are read. ● Projections are efficient. ● Any column can serve as an index.	● The selected columns need to be reconstructed after the **SELECT** statement is executed. ● Data cannot be easily inserted or updated.

Generally, if a table contains many columns (called a wide table) and its query involves only a few columns, column storage is recommended. Row storage is recommended if a table contains only a few columns and a query involves most of the fields.

Storage Model	Application Scenarios
Row storage	● Point queries (simple index-based queries that only return a few records) ● Scenarios requiring frequent addition, deletion, and modification
Column storage	● Statistical analysis queries (requiring a large number of association and grouping operations) ● Ad hoc queries (using uncertain query conditions and unable to utilize indexes to scan row-store tables)

Row-Store Tables

Row-store tables are created by default. In a row-store table, data is stored by row, that is, data in each row is stored continuously. Therefore, this storage model applies to scenarios where data needs to be updated frequently.

```
openGauss=# CREATE TABLE customer_t1
(
  state_ID   CHAR(2),
  state_NAME VARCHAR2(40),
  area_ID    NUMBER
);

--Delete the table.
openGauss=# DROP TABLE customer_t1;
```

Column-Store Tables

In a column-store table, data is stored by column, that is, data in each column is stored continuously. The I/O of data query in a single column is small, and column-store tables occupy less storage space than row-store tables. This storage model applies to scenarios where data is inserted in batches, less updated, and queried for statistical analysis. A column-store table cannot be used for point queries.

```
openGauss=# CREATE TABLE customer_t2
(
  state_ID   CHAR(2),
  state_NAME VARCHAR2(40),
  area_ID    NUMBER
)
WITH (ORIENTATION = COLUMN);

--Delete the table.
openGauss=# DROP TABLE customer_t2;
```

Selecting a Storage Model

- Update frequency

  If data is frequently updated, use a row-store table.
- Data insertion frequency

  If a small amount of data is frequently inserted each time, use a row-store table. If a large amount of data is inserted at a time, use a column-store table.
- Number of columns

  If a table is to contain many columns, use a column-store table.
- Number of columns to be queried

  If only a small number of columns (less than 50% of the total) is queried each time, use a column-store table.
- Compression ratio

  The compression ratio of a column-store table is higher than that of a row-store table. High compression ratio consumes more CPU resources.

2.5 Creating and Managing Tablespaces

Background

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

- If the initial disk partition or volume allocated to the database is full and the space cannot be logically increased, you can create and use tablespaces in other partitions until the space is reconfigured.
- Tablespaces allow the administrator to distribute data based on the schema of database objects, improving system performance.
  - A frequently used index can be placed in a disk having stable performance and high computing speed, such as a solid device.
  - A table that stores archived data and is rarely used or has low performance requirements can be placed in a disk with a slow computing speed.

- The administrator can use tablespaces to set the maximum available disk space. In this way, when a partition is shared with other data, tablespaces will not occupy excessive space in the partition.
- You can use tablespaces to control the disk space occupied by data in a database. If the usage of a disk where a tablespace resides reaches 90%, the database switches to the read-only mode. It switches back to read/write mode when the disk usage becomes less than 90%.

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

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

GaussDB provides two tablespaces: **pg_default** and **pg_global**.

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

Precautions

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

Procedure

- Create a tablespace.
  a. Run the following command to create user **jack**:
  ```
  openGauss=# CREATE USER jack IDENTIFIED BY 'xxxxxxxxx';
  ```
  If the following information is displayed, the user has been created:
  ```
  CREATE ROLE
  ```
  b. Run the following command to create a tablespace:
  ```
  openGauss=# CREATE TABLESPACE fastspace RELATIVE LOCATION 'tablespace/tablespace_1';
  ```
  If the following information is displayed, the tablespace has been created:
  ```
  CREATE TABLESPACE
  ```

fastspace is the new tablespace, and *Database node data directory***/pg_location/tablespace/tablespace_1** is an empty directory for which users have read and write permissions.

c. A database system administrator can run the following command to grant the permission of accessing the **fastspace** tablespace to user **jack**:

```
openGauss=# GRANT CREATE ON TABLESPACE fastspace TO jack;
```

If the following information is displayed, the permission has been assigned:

```
GRANT
```

- Create an object in a tablespace.

  If you have the CREATE permission on the tablespace, you can create database objects in the tablespace, such as tables and indexes.

  Take creating a table as an example:

  – Method 1: Run the following command to create a table in a specified tablespace:

  ```
  openGauss=# CREATE TABLE foo(i int) TABLESPACE fastspace;
  ```

  If the following information is displayed, the table has been created:

  ```
  CREATE TABLE
  ```

  – Method 2: Use **set default_tablespace** to set the default tablespace and then create a table:

  ```
  openGauss=# SET default_tablespace = 'fastspace';
  SET
  openGauss=# CREATE TABLE foo2(i int);
  CREATE TABLE
  ```

  In this example, **fastspace** is the default tablespace, and **foo2** is the created table.

- Use one of the following methods to query a tablespace:

  – Method 1: Check the **pg_tablespace** system catalog. Run the following command to view all the tablespaces defined by the system and users:

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

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

  ```
  openGauss=# \db
  ```

- Query the tablespace usage.

  a. Query the current usage of the tablespace.

  ```
  openGauss=# SELECT PG_TABLESPACE_SIZE('example');
  ```

  The following information is displayed:

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

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

  b. Calculate the tablespace usage.

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

- Modify a tablespace.

  Run the following command to rename tablespace **fastspace** to **fspace**:

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

- Delete a tablespace and related data.
  - Run the following command to delete user **jack**:
    ```
    openGauss=# DROP USER jack CASCADE;
    DROP ROLE
    ```
  - Run the following commands to delete tables **foo** and **foo2**:
    ```
    openGauss=# DROP TABLE foo;
    openGauss=# DROP TABLE foo2;
    ```
    If the following information is displayed, the tables have been deleted:
    ```
    DROP TABLE
    ```
  - Run the following command to delete tablespace **fspace**:
    ```
    openGauss=# DROP TABLESPACE fspace;
    DROP TABLESPACE
    ```
    NOTE

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

2.6 Creating and Managing Tables

2.6.1 Creating Tables

Background

A table is created in a database and can be saved in different databases. Tables under different schemas in a database can have the same name. Before creating a table, refer to **Planning a Storage Model**.

Procedure

Run the following statement to create a table:
```
openGauss=# CREATE TABLE customer_t1
(
    c_customer_sk             integer,
    c_customer_id             char(5),
    c_first_name              char(6),
    c_last_name               char(8)
);
```
If the following information is displayed, the table has been created:
```
CREATE TABLE
```
c_customer_sk, **c_customer_id**, **c_first_name**, and **c_last_name** are the column names of the table. **integer**, **char(5)**, **char(6)**, and **char(8)** are column name types.

2.6.2 Inserting Data to Tables

A new table contains no data. You need to insert data to the table before using it. This section describes how to insert a row or multiple rows of data using the **INSERT** command and to insert data from a specified table. For details about how to insert a large amount of data to a table in batches, see **Importing Data**.

Background

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

Table 2-2 Comparison of character set output between the client and server

Procedure	Server and Client Use Same Encoding	Server and Client Use Different Encoding
No operations are performed to the string while it is saved and read.	Your expected result is returned.	If the encoding for input and output on the client is the same, your expected result is returned.
Operations (such as executing string functions) are performed to the string while it is saved and read.	Your expected result is returned.	The result may differ from expected, depending on the operations performed to the string.
A long string is truncated while it is saved.	Your expected result is returned.	If the character sets used on the client and server have different character length, the result may differ from expected.

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

NOTE

Long strings are truncated only if **DBCOMPATIBILITY** is set to **TD** (compatible with Teradata) and **td_compatible_truncation** is set to **on**.

Run the following commands to create **table1** and **table2** to be used in the example:

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

Table 2-3 Examples

No.	Server Character Set	Client Character Set	Automatic Truncation Enabled	Example	Result	Description
1	SQL_ ASCII	UTF8	Yes	openGauss=# **INSERT INTO** *table1* **VALUES**(*1,reverse('123AA78'),reverse('123AA78'),reverse('123AA78')*);	id |a|b|c ----+------ +------+------ 1 | 87| 87| 87	A string is reversed on the server and then truncated. Because character sets used by the server and client are different, character A is displayed in multiple bytes on the server and the result is incorrect.
2	SQL_ ASCII	UTF8	Yes	openGauss=# **INSERT INTO** *table1* **VALUES**(*2,reverse('123A78'),reverse('123A78'),reverse('123A78')*);	id |a|b|c ----+------ +------+------ 2 | 873| 873| 873	A string is reversed and then automatically truncated. Therefore, the result is unexpected.
3	SQL_ ASCII	UTF8	Yes	openGauss=# **INSERT INTO** *table1* **VALUES**(*3,'87A123','87A123','87A123'*);	id | a | b | c ----+------- +-------+------- 3 | 87A1 | 87A1 | 87A1	The column length in the string type is an integer multiple of the length in client character encoding. Therefore, the result is correct after truncation.

No.	Server Character Set	Client Character Set	Automatic Truncation Enabled	Example	Result	Description
4	SQL_ASCII	UTF8	No	openGauss=# **INSERT INTO** *table2* **VALUES**(*1,reverse('123AA78'),reverse('123AA78'),reverse('123AA78')*); openGauss=# **INSERT INTO** *table2* **VALUES**(*2,reverse('123A78'),reverse('123A78'),reverse('123A78')*);	id |a|b|c ---- +------------------ --+-------- +-------- 1 | 87 321| 87 321 | 87 321 2 | 87321| 87321| 87321	Similar to the first example, multi-byte characters no longer indicate the original characters after being reversed.

Procedure

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

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

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

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

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

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

  If you do not know some of the column values, you can omit them. If no value is specified for a column, the column is set to the default value. Example:

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

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

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

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

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

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

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

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

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

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

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

NOTE

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

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

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

NOTE

If the table to be deleted is in dependent relationship with other tables, you need to delete its dependent tables first.

2.6.3 Updating Data in a Table

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

The following types of information are required when the **UPDATE** statement is used to update a row:

- Table name and column name of the data to be updated
- New column value
- Rows of the data to be updated

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

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

c_customer_sk in the table **customer_t1** must be changed from **9527** to **9876**:

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

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

For example, run the following statement to increase all the values in the **c_customer_sk** column by 100:

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

This statement does not include the **WHERE** clause, so all rows are updated. If the statement includes the **WHERE** clause, only the rows matching the clause are updated.

In the **SET** clause, the equal sign (=) indicates value setting. In the **WHERE** clause, the equal sign indicates comparison. **WHERE** may not represent an equation and can be replaced by other operators.

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

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

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

2.6.4 Viewing Data

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

2.6.5 Deleting Data from a Table

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

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

For example, to delete all the rows whose **c_customer_sk** column is **3869** from the table **customer_t1**, run the following command:

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

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

```
openGauss=# DELETE FROM customer_t1;
```

Or

```
openGauss=# TRUNCATE TABLE customer_t1;
```

NOTE

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

To delete a table, run the following command:

```
openGauss=# DROP TABLE customer_t1;
```

2.7 Querying a System Catalog

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

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

GaussDB provides the following types of system catalogs and views:

- System catalogs and views inherited from PG

  These system catalogs and views have the prefix **PG**.

- New system catalogs and views of GaussDB

  These system catalogs and views have the prefix **GS**.

Querying Database Tables

For example, you can run the following command to query the **PG_TABLES** system catalog for all tables in the **public** schema:

```
SELECT distinct(tablename) FROM pg_tables WHERE SCHEMANAME = 'public';
```

Information similar to the following is displayed:

```
     tablename
-------------------
 err_hr_staffs
 test
 err_hr_staffs_ft3
 web_returns_p1
 mig_seq_table
 films4
(6 rows)
```

Viewing Database Users

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

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

Viewing and Stopping the Running Query Statements

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

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

```
SET track_activities = on;
```

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

Step 2 View the running query statements. Run the following command to view the database names, users performing queries, query status, and the corresponding PIDs which are connected to the running query statements:

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

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

To identify only active query statements, run the following command:

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

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

```
SELECT PG_TERMINATE_BACKEND(139834759993104);
```

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

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

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

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

NOTE

If the **PG_TERMINATE_BACKEND** function is used to terminate the backend threads of the current session, the **gsql** client will be reconnected automatically rather than be logged out. The message "The connection to the server was lost. Attempting reset: Succeeded." is returned.

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

----End

2.8 Other Operations

2.8.1 Creating and Managing Schemas

Background

Schemas function as models. Schema management allows multiple users to use the same database without mutual impacts, to organize database objects as manageable logical groups, and to add third-party applications to the same schema without causing conflicts. Schema management involves creating a schema, using a schema, deleting a schema, setting a search path for a schema, and setting schema permissions.

Precautions

- GaussDB contains one or more named databases. Users and user groups are shared within the database, but their data is not shared. Any user who has connected to a server can access only the database specified in the connection request.
- A database can have one or more schemas, and a schema can contain tables and other data objects, such as data types, functions, and operators. One object name can be used in different schemas. For example, both **schema1** and **schema2** can have a table named **mytable**.
- Different from databases, schemas are not isolated. You can access the objects in a schema of the connected database if you have schema permissions. To manage schema permissions, you need to have knowledge about database permissions.
- A schema named with the **PG_** prefix cannot be created because this type of schema is reserved for the database system.
- Each time a new user is created, the system creates a schema with the same name for the new user in the current database. In other databases, such a schema needs to be manually created.
- To reference a table that is not modified with a schema name, the system uses **search_path** to find the schema that the table belongs to. **pg_temp** and **pg_catalog** are always the first two schemas to be searched no matter whether or how they are specified in **search_path**. **search_path** is a schema name list, and the first table detected in it is the target table. If no target table is found, an error will be reported. (If a table exists but the schema it belongs to is not listed in **search_path**, the search fails as well.) The first

schema in **search_path** is called "current schema". This schema is the first one to be searched. If no schema name is declared, newly created database objects are saved in this schema by default.

- Each database has a **pg_catalog** schema, which contains system catalogs and all embedded data types, functions, and operators. **pg_catalog** is a part of the search path and has the second highest search priority. It is searched after the schema of temporary tables and before other schemas specified in **search_path**. This search order ensures that database built-in objects can be found. To use a custom object that has the same name as a built-in object, you can specify the schema of the custom object.

Procedure

- Manage users and their permissions.
  - Run the following command to create a schema:
    ```
    openGauss=# CREATE SCHEMA myschema;
    ```
    If the following information is displayed, the schema named **myschema** is successfully created:
    ```
    CREATE SCHEMA
    ```
    To create or access an object in the schema, the object name in the command should consist of the schema name and the object name, which are separated by a dot (.), for example, **myschema.table**.
  - Run the following command to create a schema and specify the owner:
    ```
    openGauss=# CREATE SCHEMA myschema AUTHORIZATION omm;
    ```
    If the following information is displayed, the **myschema** schema that belongs to **omm** is created successfully:
    ```
    CREATE SCHEMA
    ```
- Use a schema.

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

  You can set **search_path** to specify the sequence of schemas in which objects are searched. The first schema listed in **search_path** will become the default schema. If no schema is specified during object creation, the object will be created in the default schema.
  - Run the following command to view **search_path**:
    ```
    openGauss=# SHOW SEARCH_PATH;
     search_path
    ----------------
    ```

```
 "$user",public
(1 row)
```

  - Run the following command to set **search_path** to **myschema** and **public** (**myschema** will be searched first):

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

- Set permissions for a schema.

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

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

  - Run the following command to revoke **PUBLIC**'s permission to create objects in the **public** schema. **public** indicates the schema and **PUBLIC** indicates all roles.

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

  - Run the following command to view the current schema:

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

  - Run the following commands to create user **jack** and grant the **usage** permission for **myschema** to the user:

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

  - Run the following command to revoke the usage permission for **myschema** from **jack**:

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

- Delete a schema.

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

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

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

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

  - Run the following command to delete user **jack**:

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

2.8.2 Creating and Managing Partitioned Tables

Background

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

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

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

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

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

Procedure

Example 1: using the default tablespace

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

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

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

If the following information is displayed, the partitioned table has been created:

```
CREATE TABLE
```

NOTE

You are advised to create a maximum of 1000 column-store partitioned tables.

- Insert data.

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

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

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

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

- Modify the row movement attributes of the partitioned table.

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

- Delete a partition.

  Delete partition **P8**.

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

- Add a partition.

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

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

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

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

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

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

- Query a partition.

  Query partition **P6**.

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

- Delete a partitioned table and its tablespaces.

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

Example 2: using a user-defined tablespace

Perform the following operations on range partitioned tables.

- Create a tablespace.

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

  If the following information is displayed, the tablespace has been created:

```
CREATE TABLESPACE
```

- Create a partitioned table.

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

PARTITION BY RANGE (ca_address_sk)
(
        PARTITION P1 VALUES LESS THAN(5000),
        PARTITION P2 VALUES LESS THAN(10000),
        PARTITION P3 VALUES LESS THAN(15000),
        PARTITION P4 VALUES LESS THAN(20000),
        PARTITION P5 VALUES LESS THAN(25000),
        PARTITION P6 VALUES LESS THAN(30000),
        PARTITION P7 VALUES LESS THAN(40000),
        PARTITION P8 VALUES LESS THAN(MAXVALUE) TABLESPACE example2
)
ENABLE ROW MOVEMENT;
```

If the following information is displayed, the partitioned table has been created:

```
CREATE TABLE
```

NOTE

You are advised to create a maximum of 1000 column-store partitioned tables.

- Insert data.

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

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

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

- Modify the row movement attributes of the partitioned table.

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

- Delete a partition.

  Delete partition **P8**.

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

- Add a partition.

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

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

- Rename a partition.
  - Rename partition **P8** to **P_9**.
    ```
    openGauss=# ALTER TABLE tpcds.web_returns_p2 RENAME PARTITION P8 TO P_9;
    ALTER TABLE
    ```
  - Rename partition **P_9** to **P8**.
    ```
    openGauss=# ALTER TABLE tpcds.web_returns_p2 RENAME PARTITION FOR (40000) TO P8;
    ALTER TABLE
    ```
- Modify the tablespace of a partition.
  - Change the tablespace of partition **P6** to **example3**.
    ```
    openGauss=#  ALTER TABLE tpcds.web_returns_p2 MOVE PARTITION P6 TABLESPACE
    example3;
    ALTER TABLE
    ```
  - Change the tablespace of partition **P4** to **example4**:
    ```
    openGauss=#  ALTER TABLE tpcds.web_returns_p2 MOVE PARTITION P4 TABLESPACE
    example4;
    ALTER TABLE
    ```
- Query a partition.

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

2.8.3 Creating and Managing Indexes

Background

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

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

- Columns that are often searched and queried. This speeds up searches.
- Columns that function as primary keys. This enforces the uniqueness of the columns and the data collation structures in organized tables.
- Columns that are often searched by range. The index helps collate data, and therefore the specified ranges are contiguous.
- Columns that often need to be collated. The index helps collate data, reducing the time for a collation query.

- Columns where the **WHERE** clause is executed frequently. This speeds up condition judgment.
- Columns that often appear after the keywords **ORDER BY**, **GROUP BY**, and **DISTINCT**.

  NOTE

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

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

Procedure

For details about how to create a partitioned table, see **Creating and Managing Partitioned Tables**.

- Create an index.
  - Create the local index **tpcds_web_returns_p2_index1** without specifying the partition name.
    ```
    openGauss=# CREATE INDEX tpcds_web_returns_p2_index1 ON tpcds.web_returns_p2
    (ca_address_id) LOCAL;
    ```
    If the following information is displayed, the index has been created:
    ```
    CREATE INDEX
    ```
  - Create the local index **tpcds_web_returns_p2_index2** with the partition name specified.
    ```
    openGauss=# CREATE INDEX tpcds_web_returns_p2_index2 ON tpcds.web_returns_p2
    (ca_address_sk) LOCAL
    (
        PARTITION web_returns_p2_P1_index,
        PARTITION web_returns_p2_P2_index TABLESPACE example3,
        PARTITION web_returns_p2_P3_index TABLESPACE example4,
        PARTITION web_returns_p2_P4_index,
        PARTITION web_returns_p2_P5_index,
        PARTITION web_returns_p2_P6_index,
        PARTITION web_returns_p2_P7_index,
        PARTITION web_returns_p2_P8_index
    ) TABLESPACE example2;
    ```
    If the following information is displayed, the index has been created:
    ```
    CREATE INDEX
    ```
  - Create the global index **tpcds_web_returns_p2_global_index** for a partitioned table.
    ```
    CREATE INDEX tpcds_web_returns_p2_global_index ON tpcds.web_returns_p2
    (ca_street_number) GLOBAL;
    ```
- Modify the tablespace of an index partition.

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

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

  If the following information is displayed, the tablespace of the index partition has been modified:

  ```
  ALTER INDEX
  ```

- Change the tablespace of index partition **web_returns_p2_P3_index** to **example2**.

  ```
  openGauss=# ALTER INDEX tpcds.tpcds_web_returns_p2_index2 MOVE PARTITION
  web_returns_p2_P3_index TABLESPACE example2;
  ```

  If the following information is displayed, the tablespace of the index partition has been modified:

  ```
  ALTER INDEX
  ```

- Rename an index partition.

  Rename the name of index partition **web_returns_p2_P8_index** to **web_returns_p2_P8_index_new**.

  ```
  openGauss=# ALTER INDEX tpcds.tpcds_web_returns_p2_index2 RENAME PARTITION
  web_returns_p2_P8_index TO web_returns_p2_P8_index_new;
  ```

  If the following information is displayed, the index partition has been renamed:

  ```
  ALTER INDEX
  ```

- Query indexes.
  - Run the following command to query all indexes defined by the system and users:

    ```
    openGauss=# SELECT RELNAME FROM PG_CLASS WHERE RELKIND='i' or RELKIND='I';
    ```

  - Run the following command to query information about a specified index:

    ```
    openGauss=# \di+ tpcds.tpcds_web_returns_p2_index2
    ```

- Delete indexes.

  ```
  openGauss=# DROP INDEX tpcds.tpcds_web_returns_p2_index1;
  openGauss=# DROP INDEX tpcds.tpcds_web_returns_p2_index2;
  ```

  If the following information is displayed, the indexes have been deleted:

  ```
  DROP INDEX
  ```

GaussDB supports four methods for creating indexes. For details, see **Table 2-4**.

NOTE

- After an index is created, the system automatically determines when to reference it. If the system determines that indexing is faster than sequenced scanning, the index will be used.
- After an index is successfully created, it must be synchronized with the associated table to ensure new data can be accurately located, which increases the data operation load. Therefore, delete unnecessary indexes periodically.

Table 2-4 Indexing method

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

- Create an ordinary table.
  ```
  openGauss=# CREATE TABLE tpcds.customer_address_bak AS TABLE tpcds.customer_address;
  INSERT 0 0
  ```
- Create an ordinary index.

  For the **tpcds.customer_address_bak** table, you need to perform the following operations frequently:
  ```
  openGauss=# SELECT ca_address_sk FROM tpcds.customer_address_bak WHERE
  ca_address_sk=14888;
  ```
  Generally, the database system needs to scan the **tpcds.customer_address_bak** table row by row to find all matched tuples. If the size of the **tpcds.customer_address_bak** table is large but only a few (possibly zero or one) of the **WHERE** conditions are met, the performance of this sequential scan is poor. If the database system uses an index to maintain the **ca_address_sk** attribute, the database system only needs to search a few tree layers for the matched tuples. This greatly improves data query performance. Furthermore, indexes can improve the update and deletion operation performance in the database.

  Run the following command to create an index:
  ```
  openGauss=# CREATE INDEX index_wr_returned_date_sk ON tpcds.customer_address_bak
  (ca_address_sk);
  CREATE INDEX
  ```
- Create a unique index.

  Create a unique index on the **SM_SHIP_MODE_SK** column in the **tpcds.ship_mode_t1** table.
  ```
  openGauss=# CREATE UNIQUE INDEX ds_ship_mode_t1_index1 ON
  tpcds.ship_mode_t1(SM_SHIP_MODE_SK);
  ```
- Create a multi-column index.

Assume you need to frequently query records with **ca_address_sk** being **5050** and **ca_street_number** smaller than **1000** in the **tpcds.customer_address_bak** table. Run the following commands:

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

Run the following command to define a composite index on **ca_address_sk** and **ca_street_number** columns:

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

- Create a partial index.

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

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

- Create an expression index.

  Assume that you need to frequently query records with **ca_street_number** smaller than **1000**, run the following command:

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

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

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

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

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

2.8.4 Creating and Managing Views

Background

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

A view is different from a base table. It is only a virtual object rather than a physical one. Only view definition is stored in the database and view data is not. The data is stored in a base table. If data in the base table changes, the data in the view changes accordingly. In this sense, a view is like a window through which users can know their interested data and data changes in the database. A view is triggered every time it is referenced.

Managing Views

- Create a view.

  Run the following command to create **MyView**:

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

  NOTE

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

- Query a view.

Run the following command to query **MyView**:

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

- View details about a specified view.

  Run the following command to view details about **MyView**:

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

- Delete a view.

  Run the following command to delete **MyView**:

```
openGauss=# DROP VIEW MyView;
DROP VIEW
```

2.8.5 Creating and Managing Sequences

Background

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

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

- Set the data type of a column to **Table 11-4**. A sequence will be automatically created by the database for this column.
- Run the **CREATE SEQUENCE** statement to create a sequence. Set the initial value of the **nextval**('*sequence_name*') function to the default value of a column.

Procedure

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

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

If the following information is displayed, the table has been created:

```
CREATE TABLE
```

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

1. Create a sequence.

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

   If the following information is displayed, the sequence has been created:

```
CREATE SEQUENCE
```

2. Set the initial value of the **nextval**('*sequence_name*') function to the default value of a column.

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

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

```
CREATE TABLE
```

3. Associate the sequence with a column.

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

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

   If the following information is displayed, the column has been specified:

   ```
   ALTER SEQUENCE
   ```

NOTE

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

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

2.9 gsql

gsql is a database connection tool provided by GaussDB and runs in the command-line interface. You can use **gsql** to connect to the server and perform operations and maintenance. In addition to basic functions for performing operations on a database, **gsql** provides multiple advanced features. For details, see **Advanced Features**.

2.9.1 gsql Overview

Basic Functions

- **Connect to the database**: By default, only the local server can be connected. To connect to a remote database, you must configure the server. For details, see "Working with Databases > Connecting to a Database > Using gsql to Connect to a Database > Remotely Connecting to a Database" in the *Developer Guide*.

  NOTE

  If the **gsql** client is used to connect to a database, the connection timeout period will be 5 minutes. If the database has not correctly set up a connection and authenticated the identity of the client within this period, **gsql** will time out and exit.

  To solve this problem, see **Troubleshooting**.

- **Run SQL statements**: Interactively entered SQL statements and specified SQL statements in a file can be executed.
- **Run meta-commands**: Meta-commands help the administrator view database object information, query cache information, format SQL output,

and connect to a new database. For details about meta-commands, see **Meta-Command Reference**.

Advanced Features

Table 2-5 lists the advanced features of **gsql**.

Table 2-5 Advanced features of **gsql**

Feature	Description

Variable	**gsql** provides a variable feature that is similar to the **shell** command of Linux. The following **\set** meta-command of **gsql** can be used to specify a variable:
	`\set varname value`
	To delete a variable specified by the **\set** command, run the following command:
	`\unset varname`
	NOTE
	● A variable is a simple name-value pair, where the value can be any characters in any length.
	● Variable names must consist of case-sensitive letters (including non-Latin letters), digits, and underscores(_).
	● If the **\set** *varname* meta-command (without the second parameter) is used, the variable is set without a value specified.
	● If the **\set** meta-command without parameters is used, values of all variables are displayed.
	For details about variable examples and descriptions, see **#en-us_topic_0294748922_en-us_topic_0059778819_en-us_topic_0058968129_li39134862**.

SQL substitution	Common SQL statements can be set to variables using the variable feature of **gsql** to simplify operations. For details about SQL substitution examples and descriptions, see **#en-us_topic_0294748922_en-us_topic_0059778819_en-us_topic_0058968129_li56915888**.
Customized prompt	Prompts of **gsql** can be customized. Prompts can be modified by changing the reserved variables of **gsql**: *PROMPT1*, *PROMPT2*, and *PROMPT3*. These variables can be set to customized values or the values predefined by **gsql**. For details, see **#en-us_topic_0294748922_en-us_topic_0059778819_en-us_topic_0058968129_li65414495**.

Feature	Description
Historical client operation records	**gsql** records client operation history. This function is enabled by specifying the **-r** parameter when a client is connected. The number of historical records can be set using the **\set** command. For example, **\set HISTSIZE 50** indicates that the number of historical records is set to **50**. **\set HISTSIZE 0** indicates that the operation history is not recorded. **NOTE** ● The default number of historical records is **32**. The maximum number of historical records is **500**. If interactively entered SQL commands contain Chinese characters, only the UTF-8 encoding environment is supported. ● For security purposes, the records containing character strings such as PASSWORD, IDENTIFIED, GS_ENCRYPT_AES128, GS_DECRYPT_AES128, GS_ENCRYPT, GS_DECRYPT, PG_CREATE_PHYSICAL_REPLICATION_SLOT_EXTERN, SECRET_ACCESS_KEY, SECRETKEY and CREATE_CREDENTIAL (case-insensitive) are considered as sensitive information and will not be recorded in historical information. This indicates that you cannot view these records in command output histories.

- Variable

  To configure a variable, run the **\set** meta-command of **gsql**. For example, to set variable *foo* to **bar**, run the following command:

  ```
  openGauss=# \set foo bar
  ```

  To reference the value of a variable, add a colon (:) before the variable. For example, to view the value of variable *foo*, run the following command:

  ```
  openGauss=# \echo :foo
  bar
  ```

  This variable quotation method is suitable for regular SQL statements and meta-commands.

  gsql pre-defines some special variables and plans the values of these variables. To ensure compatibility with later versions, do not use these variables for other purposes. For details about special variables, see **Table 2-6**.

  NOTE

  - All the special variables consist of upper-case letters, digits, and underscores (_).
  - To view the default value of a special variable, run **\echo :***varname*, for example, **\echo :***DBNAME*.

Table 2-6 Settings of special variables

Variable	How to Set	Description
DBNAME	`\set DBNAME` *dbname*	Name of the connected database. This variable is reconfigured when a database is connected.

Variable	How to Set	Description
ECHO	\set ECHO all | queries	● If this variable is set to **all**, only the query information is displayed. This has the same effect as specifying the **-a** parameter when **gsql** is used to connect to a database. ● If this variable is set to **queries**, the command line and query information are displayed. This has the same effect as specifying the **-e** parameter when **gsql** is used to connect to a database.
ECHO_HIDDEN	\set ECHO_HIDDEN on | off | noexec	When a meta-command (such as **\dg**) is used to query database information, the value of this variable determines the query behavior. ● If this variable is set to **on**, the query statements that are called by the meta-command are displayed, and then the query result is displayed. This has the same effect as specifying the **-E** parameter when **gsql** is used to connect to a database. ● If this variable is set to **off**, only the query result is displayed. ● If this variable is set to **noexec**, only the query information is displayed, and the query is not run.
ENCODING	\set ENCODING *encoding*	Character set encoding of the current client.
FETCH_COUNT	\set FETCH_COUNT *variable*	● If the value is an integer greater than **0**, for example, *n*, *n* lines will be selected from the result set to the cache and displayed on the screen when the **SELECT** statement is run. ● If this variable is not set or set to a value less than or equal to **0**, all results are selected at a time to the cache when the **SELECT** statement is run. **NOTE** Setting this variable to a proper value reduces memory usage. Generally, the proper value ranges from **100** to **1000**.

Variable	How to Set	Description
HISTCONTROL	`\set HISTCONTROL ignorespace | ignoredups | ignoreboth | none`	• **ignorespace**: A line started with a space is not written to the historical record. • **ignoredups**: A line that exists in the historical record is not written to the historical record. • **ignoreboth**, **none**, or other values: All the lines read in interaction mode are saved in the historical record. **NOTE** **none** indicates that **HISTCONTROL** is not configured.
HISTFILE	`\set HISTFILE` *filename*	This file is used to store historical records. The default value is **~/.bash_history**.
HISTSIZE	`\set HISTSIZE` *size*	Number of commands in the history command. The default value is **500**.
HOST	`\set HOST` *hostname*	Name of the connected database host.
IGNOREEOF	`\set IGNOREEOF` *variable*	• If this variable is set to a number, for example, **10**, the first nine EOF characters (generally **Ctrl**+**C**) entered in **gsql** are ignored and the **gsql** program exits when the tenth **Ctrl**+**C** is entered. • If this variable is set to a non-numeric value, the default value is **10**. • If this variable is deleted, **gsql** exits when an EOF is entered.
LASTOID	`\set LASTOID` *oid*	Last OID, which is the value returned by an **INSERT** or **lo_import** command. This variable is valid only before the output of the next SQL statement is displayed.

Variable	How to Set	Description
ON_ERROR_ROLLBACK	\set ON_ERROR_ROLLBACK on | interactive | off	• If the value is **on**, an error that may occur in a statement in a transaction block is ignored and the transaction continues. • If the value is **interactive**, the error is ignored only in an interactive session. • If the value is **off** (the default value), the error triggers the rollback of the transaction block. In **on_error_rollback-on** mode, a **SAVEPOINT** is configured before each statement of a transaction block, and an error triggers the rollback of the transaction block.
ON_ERROR_STOP	\set ON_ERROR_STOP on | off	• **on**: specifies that the execution stops if an error occurs. In interactive mode, **gsql** returns the output of executed commands immediately. • **off** (default value): specifies that an error that occurs during the execution is ignored, and the execution continues.
PORT	\set PORT *port*	Port number of a connected database.
USER	\set USER *username*	Database user that is currently used for connection.
VERBOSITY	\set VERBOSITY terse | default | verbose	This variable can be set to **terse**, **default**, or **verbose** to control redundant lines of error reports. • **terse**: Only critical and major error texts and text locations are returned (which is suitable for single-line error information). • **default**: Critical and major error texts and text locations, error details, and error messages (possibly involving multiple lines) are all returned. • **verbose**: All error information is returned.

- SQL substitution

  gsql, like a parameter of a meta-command, provides a key feature that enables you to substitute a standard SQL statement for a **gsql** variable. **gsql** also provides a new alias or identifier for the variable. To replace the value of a variable using the SQL substitution method, add a colon (:) in front of the variable. For example:

```
openGauss=# \set foo 'HR.areaS'
openGauss=# select * from :foo;
 area_id |       area_name
---------+------------------------
       4 | Middle East and Africa
       3 | Asia
       1 | Europe
       2 | Americas
(4 rows)
```

The above command queries the **HR.areaS** table.

NOTICE

The value of a variable is copied character by character, and even an asymmetric quote mark or backslash (\) is copied. Therefore, the input content must be meaningful.

- Prompt

  The **gsql** prompt can be set using the three variables in **Table 2-7**. These variables consist of characters and special escape characters.

Table 2-7 Prompt variables

Variable	Description	Example

PROMPT1	Normal prompt used when **gsql** requests a new command.
	The default value of *PROMPT1* is:
	`%/%R%#`
		PROMPT1 can be used to change the prompt.
		• Change the prompt to **[local]**:
		`openGauss=> \set PROMPT1 %M`
		`[local:/tmp/gaussdba_mppdb]`
		• Change the prompt to **name**:
		`openGauss=> \set PROMPT1 name`
		`name`
		• Change the prompt to **=**:
		`openGauss=> \set PROMPT1 %R`
		`=`

PROMPT2	Prompt displayed when more command input is expected. For example, it is expected if a command is not terminated with a semicolon (;) or a quote (") is not closed.	*PROMPT2* can be used to display the prompt.
		`openGauss=# \set PROMPT2 TEST`
		`openGauss=# select * from HR.areaS TEST;`
		` area_id |       area_name`
		`---------+--------------------`
		`       1 | Europe`
		`       2 | Americas`
		`       4 | Middle East and Africa`
		`       3 | Asia`
		`(4 rows))`

Variable	Description	Example

PROMPT3	Prompt displayed when the **COPY** statement (such as **COPY FROM STDIN**) is run and data input is expected.	*PROMPT3* can be used to display the **COPY** prompt.
		`openGauss=# \set PROMPT3 '>>>>'`
		`openGauss=# copy HR.areaS from STDIN;`
		`Enter data to be copied followed by a newline.`
		`End with a backslash and a period on a line by itself.`
		`>>>>1 aa`
		`>>>>2 bb`
		`>>>>\.`

The value of the selected prompt variable is printed literally. However, a value containing a percent sign (%) is replaced by the predefined contents depending on the character following the percent sign (%). For details about the defined substitutions, see **Table 2-8**.

Table 2-8 Defined substitutions

Symbol	Description
%M	Full host name (with domain name). It can be [local] if the connection is over a Unix domain socket, or [local:*/dir/name*], if the Unix domain socket is not at the compiled in default location.
%m	Host name truncated at the first dot. It is [local] if the connection is over a Unix domain socket.
%>	Number of the port that the host is listening on.
%n	User name of a database session.
%/	Name of the current database.
%~	Similar to **%/**. If the database is the default database, the output is a tilde (~).
%#	If the session user is a database system administrator, use **#**. Otherwise, use **>**.
%R	• **=** is normally used for *PROMPT1*, but **^** is used in single-line mode and **!** is used if the session is disconnected from the database (which may occur if **\connect** fails). • For *PROMPT2*, the sequence is replaced by a hyphen (-), asterisk (*), single quotation mark ('), double quotation mark ("), or dollar sign ($), depending on whether gsql is waiting for more input, or the query is not terminated, or the query is in the /* ... */ the comment, quotation mark, or dollar sign extension.

Symbol	Description
%x	Transaction state. ● An empty string when it is not in a transaction block ● An asterisk (*) when it is in a transaction block ● An exclamation mark (!) when it is in a failed transaction block ● A question mark (?) when the transaction status is indeterminate (for example, because there is no connection).
%digits	Is replaced with the character with the specified byte.
%:name	Value of the gsql variable name.
%comma nd	Command output, similar to ordinary "back-tick" ("^") substitution.
%[. . . %]	Prompts can contain terminal control characters which can change the color, background, or style of the prompt text, or change the title of the terminal window. For example: openGauss=> \set PROMPT1 '%[%033[1;33;40m%]%n@%/%R %[%033[0m%]%#' The output is a boldfaced (1;) yellow-on-black (33;40) prompt on VT100-compatible and color-capable terminals.

Environment Variables

Table 2-9 Environment variables related to **gsql**

Name	Description
COLUMNS	If **\set columns** is set to **0**, this parameter controls the width of the wrapped format. This width determines whether the width output mode is changed to a vertical bar format in automatic expansion mode.
PAGER	If the query result cannot be displayed within one page, the query result will be redirected to the command. You can use the **\pset** command to disable the pager. Typically, the **more** or **less** command is used for viewing the query result page by page. The default value is platform-associated. **NOTE** Display of the **less** command is affected by the *LC_CTYPE* environmental variable.
PSQL_EDITOR	The **\e** and **\ef** commands use the editor specified by the environment variables. Variables are checked according to the list sequence. The default editor on Unix is vi.
EDITOR	

Name	Description
VISUAL	

PSQL_EDITOR_LINENUMBER_ARG	When the **\e** or **\ef** command is used with a line number parameter, this variable specifies the command-line parameter used to pass the starting line number to the editor. For editors, such as Emacs or vi, this is a plus sign. A space is added behind the value of the variable if whitespace is required between the option name and the line number. For example:
	`PSQL_EDITOR_LINENUMBER_ARG = '+'`
	`PSQL_EDITOR_LINENUMBER_ARG='--line '`
	A plus sign (+) is used by default on Unix.

PSQLRC	Specifies the location of the user's **.gsqlrc** file.
SHELL	Has the same effect as the **\!** command.
TMPDIR	Specifies the directory for storing temporary files. The default value is **/tmp**.

2.9.2 Usage Guidelines

Prerequisites

The user has the permission to access the database.

Background

You can use the **gsql** command to connect to the local database or remote database.

Procedure

Step 1 Connect to the GaussDB server using the **gsql** tool.

The **gsql** tool uses the **-d** parameter to specify the target database name, the **-U** parameter to specify the database username, the **-h** parameter to specify the host name, and the **-p** parameter to specify the port number.

NOTE

If the database name is not specified, the default database name generated during initialization will be used. If the database username is not specified, the OS user is used as the database username by default. If a variable does not belong to any parameter (such as **-d** and **-U**), and **-d** is not specified, the variable will be used as the database username. If **-d** is specified but **-U** is not specified, the variable will be used as the database username.

Example: Connect to the port 8000 of the remote postgres database as user **jack**.

```
gsql -h 10.180.123.163 -d postgres -U jack -p 8000
```

In a centralized database instance, when connecting to the primary DN, you can use commas (,) to separate the IP addresses of DNs and add them to the end of -

h. **gsql** connects to each IP address in sequence to check whether the current DN is the primary DN. If no, **gsql** disconnects from the current IP address and attempts to connect to the next IP address until the primary DN is found.

```
gsql -h 10.180.123.163,10.180.123.164,10.180.123.165 -d postgres -U jack -p 8000
```

Example 3: *postgres* and *omm* do not belong to any parameter, and they are used as the database name and the username, respectively.

```
gsql postgres omm -p 8000
```

Equals

```
gsql -d postgres -U omm -p 8000
```

For details about the **gsql** parameters, see **Command Reference**.

Step 2 Run the SQL statement.

The following takes creating database **human_staff** as an example:

```
CREATE DATABASE human_staff;
CREATE DATABASE
```

Typically, input lines end when a command-terminating semicolon is reached. If the command was sent and run without any error, the command output is displayed on the screen.

Step 3 Execute gsql meta-commands.

The following takes all GaussDB databases and description information as an example:

```
openGauss=# \l
                                List of databases
      Name      |  Owner   | Encoding  | Collate | Ctype |   Access privileges
----------------+----------+-----------+---------+-------+-----------------------
 human_resource | omm | SQL_ASCII | C       | C     |
 postgres       | omm | SQL_ASCII | C       | C     |
 template0      | omm | SQL_ASCII | C       | C     | =c/omm         +
                |          |           |         |       | omm=CTc/omm
 template1      | omm | SQL_ASCII | C       | C     | =c/omm          +
                |          |           |         |       | omm=CTc/omm
 human_staff    | omm | SQL_ASCII | C       | C     |
(5 rows)
```

For more gsql meta-commands, see **Meta-Command Reference**.

----End

Example

The example shows how to spread a command over several lines of input. Note the prompt change:

```
openGauss=# CREATE TABLE HR.areaS(
openGauss(# area_ID   NUMBER,
openGauss(# area_NAME VARCHAR2(25)
openGauss-# )tablespace EXAMPLE;
CREATE TABLE
```

Query the table definition:

```
openGauss=# \d HR.areaS
               Table "hr.areas"
  Column   |          Type          | Modifiers
```

```
-----------+-----------------------+-----------
 area_id   | numeric               | not null
 area_name | character varying(25) |
```

Insert four lines of data into **HR.areaS**.

```
openGauss=# INSERT INTO HR.areaS (area_ID, area_NAME) VALUES (1, 'Europe');
INSERT 0 1
openGauss=# INSERT INTO HR.areaS (area_ID, area_NAME) VALUES (2, 'Americas');
INSERT 0 1
openGauss=# INSERT INTO HR.areaS (area_ID, area_NAME) VALUES (3, 'Asia');
INSERT 0 1
openGauss=# INSERT INTO HR.areaS (area_ID, area_NAME) VALUES (4, 'Middle East and Africa');
INSERT 0 1
```

Change the prompt.

```
openGauss=# \set PROMPT1 '%n@%m %~%R%#'
omm@[local] openGauss=#
```

Query the table.

```
omm@[local] openGauss=# SELECT * FROM HR.areaS;
 area_id |       area_name
---------+------------------------
       1 | Europe
       4 | Middle East and Africa
       2 | Americas
       3 | Asia
(4 rows)
```

Use the **\pset** command to display the table in different ways:

```
omm@[local] openGauss=# \pset border 2
Border style is 2.
omm@[local] openGauss=# SELECT * FROM HR.areaS;
+---------+------------------------+
| area_id |       area_name        |
+---------+------------------------+
|       1 | Europe                 |
|       2 | Americas               |
|       3 | Asia                   |
|       4 | Middle East and Africa |
+---------+------------------------+
(4 rows)
omm@[local] openGauss=# \pset border 0
Border style is 0.
omm@[local] openGauss=# SELECT * FROM HR.areaS;
area_id       area_name
------- ----------------------
      1 Europe
      2 Americas
      3 Asia
      4 Middle East and Africa
(4 rows)
```

Use the meta-command:

```
omm@[local] openGauss=# \a \t \x
Output format is unaligned.
Showing only tuples.
Expanded display is on.
omm@[local] openGauss=# SELECT * FROM HR.areaS;
area_id|2
area_name|Americas

area_id|1
area_name|Europe

area_id|4
```

```
area_name|Middle East and Africa

area_id|3
area_name|Asia
omm@[local] openGauss=#
```

2.9.3 Help Information

Procedure

- When a database is being connected, run the following commands to obtain the help information:

  ```
  gsql --help
  ```

  The following help information is displayed:

  ```
  ......
  Usage:
    gsql [OPTION]... [DBNAME [USERNAME]]

  General options:
    -c, --command=COMMAND    run only single command (SQL or internal) and exit
    -d, --dbname=DBNAME      database name to connect to (default: "omm")
    -f, --file=FILENAME      execute commands from file, then exit
  ......
  ```

- After the database is connected, run the following commands to obtain the help information:

  ```
  help
  ```

  The following help information is displayed:

  ```
  You are using gsql, the command-line interface to gaussdb.
  Type:  \copyright for distribution terms
         \h for help with SQL commands
         \? for help with gsql commands
         \g or terminate with semicolon to execute query
         \q to quit
  ```

Examples

Step 1 Run the following command to connect to the database:

```
gsql -d postgres -p 8000
```

postgres is the name of the database to be connected, and 8000 is the port number of the database primary node.

If information similar to the following is displayed, the connection succeeds:

```
gsql ((GaussDB Kernel VxxxRxxxCxx build 290d125f) compiled at 2020-05-08 02:59:43 commit 2143 last mr 131)
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.
```

Step 2 View the **gsql** help information. For details, see **Table 2-10**.

Table 2-10 gsql online help

Description	Example
View copyright information	\copyright

<table>
<tr><th>Description</th><th>Example</th></tr>
<tr><td>View help information about SQL statements supported by GaussDB.</td><td>View help information about SQL statements supported by GaussDB.
For example, view all SQL statements supported by GaussDB.<pre>
openGauss=# \h
Available help:
  ABORT
  ALTER AGGREGATE
  ALTER APP WORKLOAD GROUP
... ...
</pre>For example, view parameters of the <b>CREATE DATABASE</b> command:<pre>
openGauss=# \help CREATE DATABASE
Command: CREATE DATABASE
Description: create a new database
Syntax:
CREATE DATABASE database_name
    [[WITH] {[OWNER [=] user_name]|
           [TEMPLATE [=] template]|
           [ENCODING [=] encoding]|
           [LC_COLLATE [=] lc_collate]|
           [LC_CTYPE [=] lc_ctype]|
           [DBCOMPATIBILITY [=] compatibility_type]|
           [TABLESPACE [=] tablespace_name]|
           [CONNECTION LIMIT [=] connlimit]}[...]];
</pre></td></tr>
<tr><td>View help information about <b>gsql</b> commands</td><td>For example, view commands supported by <b>gsql</b>.<pre>
openGauss=# \?
General
  \copyright   show openGauss usage and distribution terms
  \g [FILE] or ;   execute query (and send results to file or |pipe)
  \h(\help) [NAME]   help on syntax of SQL commands, * for all
commands
  \q   quit gsql
... ...
</pre></td></tr>
</table>

----End

2.9.4 Command Reference

Table 2-11, **Table 2-12**, **Table 2-13** and **Table 2-14** list the **gsql** parameters.

Table 2-11 Common Parameters

Parameter	Description	Value Range
-c, --command=COMMAND	Specifies that **gsql** is to run a string command and then exit.	-

Parameter	Description	Value Range
-d, --dbname=DBNAME	Specifies the name of the database to connect to. In addition, **gsql** allows you to use extended database names, that is, connection strings in the format of **'postgres[ql]://[user[:password]@][netloc][:port][, ...][/dbname][?param1=value1&...]'** or **'[key=value] [...]'** as database names. **gsql** parses connection information from the connection strings and preferentially uses the information.	Character string
-f, --file=FILENAME	Specifies that files are used as the command source instead of interactively-entered commands. After the files are processed, **gsql** exits. If *FILENAME* is a hyphen (-), then standard input is read.	An absolute path or relative path that meets the OS path naming convention
-l, --list	Lists all available databases and then exits.	-
-v, --set, --variable=NAME=VALUE	Sets the gsql variable *NAME* to *VALUE*. For details about variable examples and descriptions, see **#en-us_topic_0294748922_en-us_topic_0059778819_en-us_topic_0058968129_li39134862**.	-
-X, --no-gsqlrc	Does not read the startup file (including system-wide **gsqlrc** file and the user's **~/.gsqlrc** file). **NOTE** The startup file is **~/.gsqlrc** by default or it can be specified by the environment variable *PSQLRC*.	-
-1 ("one"), --single-transaction	When **gsql** uses the **-f** parameter to execute a script, **START TRANSACTION** and **COMMIT** are added to the start and end of the script, respectively, so that the script is executed as one transaction. This ensures that the script is executed successfully. If the script cannot be executed, the script is invalid. **NOTE** If the script has used **START TRANSACTION**, **COMMIT**, or **ROLLBACK**, this parameter is invalid.	-
-?, --help	Displays help information about **gsql** command line parameters, and exits.	-

Parameter	Description	Value Range
-V, --version	Prints the **gsql** version and exits.	-

Table 2-12 Input and output parameters

Parameter	Description	Value Range
-a, --echo-all	Prints all input lines to standard output as they are read. **CAUTION** When this parameter is used in some SQL statements, sensitive information, such as user passwords, may be disclosed. Use this parameter with caution.	-
-e, --echo-queries	Copies all SQL statements sent to the server to standard output as well. **CAUTION** When this parameter is used in some SQL statements, sensitive information, such as user passwords, may be disclosed. Use this parameter with caution.	-
-E, --echo-hidden	Echoes the actual queries generated by **\d** and other backslash commands.	-
-k, --with-key=KEY	Uses **gsql** to decrypt imported encrypted files. **NOTICE** ● For key characters, such as the single quotation mark (') or double quotation mark (") in shell commands, Linux shell checks whether the input single quotation mark (') or double quotation mark (") matches. If no match is found, Linux shell does not enter the **gsql** program until input is complete. ● Stored procedures and functions cannot be decrypted and imported.	-
-L, --log-file=FILENAME	Writes normal output destination and all query output into the **FILENAME** file. **CAUTION** ● When this parameter is used in some SQL statements, sensitive information, such as user passwords, may be disclosed. Use this parameter with caution. ● This parameter retains only the query result in the corresponding file, so that the result can be easily found and parsed by other invokers (for example, automatic O&M scripts). Logs about **gsql** operation are not retained.	An absolute path or relative path that meets the OS path naming convention

Parameter	Description	Value Range
-m, --maintenance	Allows connections to the database during two-phase transaction recovery. **NOTE** The parameter is for engineers only. When this parameter is used, **gsql** can be connected to the standby server to check data consistency between the primary server and standby server.	-
-n, --no-libedit	Closes the command line editing.	-
-o, --output=FILENAME	Puts all query output into the **FILENAME** file.	An absolute path or relative path that meets the OS path naming convention
-q, --quiet	Specifies the quiet mode and no additional information will be printed.	By default, **gsql** displays various information.
-s, --single-step	Runs in single-step mode. This indicates that the user is prompted before each command is sent to the server. This parameter can also be used for canceling execution. This parameter can be used to debug scripts. **CAUTION** When this parameter is used in some SQL statements, sensitive information, such as user passwords, may be disclosed. Use this parameter with caution.	-
-S, --single-line	Runs in single-row mode where a new line terminates an SQL statement in the same manner as a semicolon does.	-
-C, --enable-client-encryption	When **-C** is used to connect to a local or remote database, you can use this parameter to enable the encrypted database function.	-

Table 2-13 Output format parameters

Parameter	Description	Value Range
-A, --no-align	Switches to unaligned output mode.	The default output mode is aligned.
-F, --field-separator=STRING	Specifies the field separator. The default is the vertical bar (|).	-
-H, --html	Turns on the HTML tabular output.	-
-P, --pset=VAR[=ARG]	Specifies the print option in the \pset format in the command line. **NOTE** The equal sign (=), instead of the space, is used to separate the name and value. For example, enter **-P format=latex** to set the output format to **LaTeX**.	-
-R, --record-separator=STRING	Specifies the record separators.	-
-r	Enables the editing mode on the client.	This function is disabled by default.
-t, --tuples-only	Prints only tuples.	-
-T, --table-attr=TEXT	Specifies options to be placed within the HTML table tag. Works with the **-H,--html** parameter to specify the output to the HTML format.	-
-x, --expanded	Turns on the expanded table formatting mode.	-
-z, --field-separator-zero	Sets the field separator in the unaligned output mode to be blank. Use this parameter with the **-A, --no-align** parameter to switch to unaligned output mode.	-
-0, --record-separator-zero	Sets the record separator in the unaligned output mode to be blank. Use this parameter with the **-A, --no-align** parameter to switch to unaligned output mode.	-
-2, --pipeline	Uses a pipe to transmit the password. This parameter cannot be used on devices and must be used together with the **-c** or **-f** parameter.	-

Parameter	Description	Value Range
-g,	Prints all SQL statements from a file.	-

Table 2-14 Connection parameters

Parameter	Description	Value Range
-h, --host=HOSTNAME	Specifies the host name of the machine on which the server is running or the directory for the Unix-domain socket.	If the host name is omitted, **gsql** connects to the server of the local host over the Unix domain socket or over TCP/IP to connect to local host without the Unix domain socket.
-p, --port=PORT	Specifies the port number of the database server. You can modify the default port number using the **-p, --port=PORT** parameter.	The default value is 8000.
-U, --username=USERNAME	Specifies the user that connects to the database. **NOTE** ● If a user is specified to access a database using this parameter, a user password must be provided together for identity verification. You can enter the password interactively or use the **-W** parameter to specify a password. ● To connect to a database, add an escape character before any dollar sign ($) in the username.	String. The default username is the same as the current OS username.

Parameter	Description	Value Range
-W, --password=PASSWORD	Specifies a password when the **-U** parameter is used to connect to the local database or a remote database. **NOTE** ● When the server where the primary database node is located connects to the local primary database node, the trust connection is used by default and this parameter is ignored. ● To connect to a database, add an escape character before any backslash (\) or back quote (`) in the password. ● If this parameter is not specified but database connection requires your password, you will be prompted to enter your password in interactive mode. The maximum length of the password is 999 bytes, which is restricted by the maximum value of **password_max_length**.	String

2.9.5 Meta-Command Reference

This section describes meta-commands provided by **gsql** after the GaussDB database CLI tool is used to connect to a database. A **gsql** meta-command can be anything that you enter in **gsql** and begins with an unquoted backslash (\).

Precautions

- The format of the **gsql** meta-command is a backslash(\) followed by a command verb, and then a parameter. The parameters are separated from the command verb and from each other by any number of whitespace characters.
- To include whitespace in a parameter, you can quote it with single quotation marks (''). To include single quotation marks in a parameter, add a backslash in front of it. Anything contained in single quotation marks is furthermore subject to C-like substitutions for \n (new line), \t (tab), \b (backspace), \r (carriage return), \f (form feed), \digits (octal), and \xdigits (hexadecimal).
- Within a parameter, text enclosed in double quotation marks ("") is taken as a command line input to the shell. The command output (with any trailing newline removed) is taken as a parameter.
- If an unquoted argument begins with a colon (:), the parameter is taken as a **gsql** variable and the value of the variable is used as the parameter value instead.
- Some commands take an SQL identifier (such as a table name) as a parameter. These parameters follow the SQL syntax rules: Unquoted letters are forcibly converted to lowercase letters, while double quotation marks ("") protect letters from case conversion and allow incorporation of whitespace into the identifier. Within double quotation marks, paired double quotation marks reduce to a single double quotation mark in the result name. For

example, **FOO"BAR"BAZ** is interpreted as **fooBARbaz**, and **"Aweird""name"** becomes **A weird"name**.

- Parameter parsing stops when another unquoted backslash appears. An unquoted backslash is taken as the beginning of a new meta-command. The special sequence \\ (two backslashes) marks the end of parameters and continues parsing SQL statements if any. In this way, SQL statements and **gsql** commands can be freely mixed in a line. However, the parameters of a meta-command cannot continue beyond the end of a line in any situations.

Meta-command

For details about meta-commands, see **Table 2-15**, **Table 2-16**, **Table 2-17**, **Table 2-18**, **Table 2-20**, **Table 2-22**, **Table 2-23**, **Table 2-24** and **Table 2-26**.

NOTICE

FILE mentioned in the following commands indicates a file path. This path can be an absolute path such as **/home/gauss/file.txt** or a relative path, such as **file.txt**. By default, a **file.txt** is created in the path where the user runs **gsql** commands.

Table 2-15 General meta-commands

Parameter	Description	Value Range
\copyright	Displays the version and copyright information of GaussDB.	-
\g [FILE] or ;	Executes the query (and sends the result to a file or pipe).	-
\h(\help) [NAME]	Provides syntax help for a specified SQL statement.	If the name is not specified, then **gsql** will list all the commands for which syntax help is available. If the name is an asterisk (*), syntax help on all SQL statements is displayed.

Parameter	Description	Value Range
\parallel [on [num]|off]	Controls the parallel execution function. ● **on**: The function is enabled and the maximum number of concurrently executed tasks is **num**. ● **off**: The function is disabled. **NOTE** ● Parallel execution is not allowed in a running transaction and a transaction is not allowed to be started during parallel execution. ● Parallel execution of **\d** meta-commands is not allowed. ● If SELECT statements are run concurrently, customers can accept the problem that the return results are displayed randomly but they cannot accept it if a core dump or process response failure occurs. ● The SET statements are not allowed in concurrent tasks because they may cause unexpected results. ● Temporary tables cannot be created in parallel. If temporary tables are required, create them before parallel execution is enabled, and use them only in the parallel execution. Temporary tables cannot be created in parallel execution. ● When **\parallel** is executed, *num* independent **gsql** processes can be connected to the database server. ● The duration of all jobs in **\parallel** cannot exceed the value of **session_timeout**. Otherwise, the connection may be interrupted during concurrent execution. ● One or more commands following **\parallel on** will be executed only after **\parallel off** is executed. Therefore, **\parallel on** must be followed by **\parallel off**. Otherwise, the commands following **\parallel on** cannot be executed.	The default value of *num* is **1024**. **NOTICE** ● The maximum number of connections allowed by the server is determined based on **max_connection** and the number of current connections. ● Configure the value of *num* based on the allowed number of connections.
\q	Exits the **gsql** program. In a script file, this command is run only when a script terminates.	-

Table 2-16 Buffer query meta-commands

Parameter	Description
\e [FILE] [LINE]	Edits the query buffer or file using an external editor.
\ef [FUNCNAME [LINE]]	Edits the function definition using an external editor. If **LINE** is specified, the cursor will point to the specified line of the function body.
\p	Prints the current query buffer to the standard output.
\r	Resets (clears) the query buffer.
\w FILE	Outputs the current query buffer to a file.

Table 2-17 Input and output meta-commands

Parameter	Description
\copy { table [(column_list)] | (query) } { from | to } { filename | stdin | stdout | pstdin | pstdout } [with] [binary] [oids] [delimiter [as] 'character'] [null [as] 'string'] [csv [header] [quote [as] 'character'] [escape [as] 'character'] [force quote column_list | *] [force not null column_list]] [parallel integer]	After logging in to the database on any **gsql** client, you can import and export data. This is an operation of running the **SQL COPY** command, but not the server that reads or writes data to a specified file. Instead, data is transferred between the server and the local file system. This means that the accessibility and permissions of the file are the permissions of the local user rather than the server. The initial database user permission is not required. **NOTE** **\COPY** applies only to small-scale data import in good format. It neither preprocesses invalid characters nor provides error tolerance. **GDS** or **COPY** is preferred for data import. **\COPY** specifies the number of clients to import data to implement parallel import of data files. Currently, the value ranges from 1 to 8. The parallel import using **\COPY** has the following constraints: Parallel import of temporary tables is not supported. Parallel import within transactions is not supported. Parallel import of binary files is not supported. Parallel import of data encrypted using AES-128 is not supported. The COPY option contains EOL. In these cases, even if the parallel parameter is specified, a non-parallel process is performed.
\echo [STRING]	Writes a character string to the standard output.

Parameter	Description
\i FILE	Reads content from *FILE* and uses them as the input for a query.
\i+ FILE KEY	Runs commands in an encrypted file.
\ir FILE	Is similar to **\i**, but resolves relative path names differently.
\ir+ FILE KEY	Is similar to **\i+**, but resolves relative path names differently.
\o [FILE]	Saves all query results to a file.
\qecho [STRING]	Prints a character string to the query result output.

NOTE

In **Table 2-18**, **S** indicates that the system object is displayed, and **+** indicates that additional object descriptions are displayed. **PATTERN** specifies the name of the object to be displayed.

Table 2-18 Information display meta-commands

Parameter	Description	Value Range	Example
\d[S+]	Lists all tables, views, and sequences of all schemas in **search_path**. When objects with the same name exist in different schemas in **search_path**, only the object in the schema that ranks first in **search_path** is displayed.	-	Lists all tables, views, and sequences of all schemas in **search_path**. `openGauss=# \d`
\d[S+] NAME	Lists the structure of specified tables, views, and indexes.	-	Lists the structure of table **a**. `openGauss=# \dtable+ a`
\d+ [PATTERN]	Lists all tables, views, and indexes.	If **PATTERN** is specified, only tables, views, and indexes whose names match **PATTERN** are shown.	Lists all tables, views, and indexes whose names start with **f**. `openGauss=# \d+ f*`

Parameter	Description	Value Range	Example
\da[S] [PATTERN]	Lists all available aggregate functions, together with their return value types and the data types.	If **PATTERN** is specified, only aggregate functions whose names match **PATTERN** are displayed.	Lists all available aggregate functions whose names start with **f**, together with their return value types and the data types. `openGauss=# \da f*`
\db[+] [PATTERN]	Lists all available tablespaces.	If **PATTERN** is specified, only tablespaces whose names match **PATTERN** are shown.	Lists all available tablespaces whose names start with **p**. `openGauss=# \db p*`
\dc[S+] [PATTERN]	Lists all available conversions between character sets.	If **PATTERN** is specified, only conversions whose names match **PATTERN** are displayed.	Lists all available conversions between character sets. `openGauss=# \dc *`
\dC[+] [PATTERN]	Lists all available type conversions. **PATTERN** must be the actual type name and cannot be an alias.	If **PATTERN** is specified, only conversions whose names match **PATTERN** are displayed.	Lists all type conversions whose pattern names start with **c**. `openGauss=# \dC c*`
\dd[S] [PATTERN]	Lists descriptions about objects matching **PATTERN**.	If no parameter is specified, all visible objects are displayed. The objects include aggregations, functions, operators, types, relations (table, view, index, sequence, and large object), and rules.	Lists all visible objects. `openGauss=# \dd`
\ddp [PATTERN]	Lists all default permissions.	If **PATTERN** is specified, only permissions whose names match **PATTERN** are displayed.	Lists all default permissions. `openGauss=# \ddp`

Parameter	Description	Value Range	Example
\dD[S+] [PATTERN]	Lists all available domains.	If **PATTERN** is specified, only domains whose names match **PATTERN** are shown.	Lists all available domains. `openGauss=# \dD`
\ded[+] [PATTERN]	Lists all data sources.	If **PATTERN** is specified, only objects whose names match **PATTERN** are shown.	Lists all data sources. `openGauss=# \ded`
\det[+] [PATTERN]	Lists all external tables.	If **PATTERN** is specified, only tables whose names match **PATTERN** are shown.	Lists all external tables. `openGauss=# \det`
\des[+] [PATTERN]	Lists all external servers.	If **PATTERN** is specified, only servers whose names match **PATTERN** are shown.	Lists all external servers. `openGauss=# \des`
\deu[+] [PATTERN]	Lists all user mappings.	If **PATTERN** is specified, only information whose name matches **PATTERN** is displayed.	Lists all user mappings. `openGauss=#\deu`
\dew[+] [PATTERN]	Lists all encapsulated external data.	If **PATTERN** is specified, only data whose name matches **PATTERN** is displayed.	Lists all encapsulated external data. `openGauss=# \dew`

Parameter	Description	Value Range	Example
\df[antw][S+] [PATTERN]	Lists all available functions, together with their parameters and return types. **a** indicates an aggregate function, **n** indicates a common function, **t** indicates a trigger, and **w** indicates a window function.	If **PATTERN** is specified, only functions whose names match **PATTERN** are displayed.	Lists all available functions, together with their parameters and return types. `openGauss=# \df`
\dF[+] [PATTERN]	Lists all text search configurations.	If **PATTERN** is specified, only configurations whose names match **PATTERN** are displayed.	Lists all text search configurations. `openGauss=# \dF+`
\dFd[+] [PATTERN]	Lists all text search dictionaries.	If **PATTERN** is specified, only dictionaries whose names match **PATTERN** are displayed.	Lists all text search dictionaries. `openGauss=# \dFd`
\dFp[+] [PATTERN]	Lists all text search parsers.	If **PATTERN** is specified, only analyzers whose names match **PATTERN** are displayed.	Lists all text search parsers. `openGauss=# \dFp`
\dFt[+] [PATTERN]	Lists all text search templates.	If **PATTERN** is specified, only templates whose names match **PATTERN** are displayed.	Lists all text search templates. `openGauss=# \dFt`
\dg[+] [PATTERN]	Lists all database roles. **NOTE** Since the concepts of "users" and "groups" have been unified into "roles", this command is equivalent to **\du**. To be compatible with earlier versions, two commands are reserved.	If **PATTERN** is specified, only roles whose names match **PATTERN** are displayed.	Lists all database roles whose names start with **j** and end with **e**. `openGauss=# \dg j?e`
\dl	Specifies an alias for **\lo_list**, which shows a list of large objects.	-	Lists all large objects. `openGauss=# \dl`

Parameter	Description	Value Range	Example
\dL[S+] [PATTERN]	Lists available procedural languages.	If **PATTERN** is specified, only languages whose names match **PATTERN** are listed.	Lists available procedural languages. `openGauss=# \dL`
\dm[S+] [PATTERN]	Lists materialized views.	If **PATTERN** is specified, only materialized views whose names match **PATTERN** are displayed.	Lists materialized views. `openGauss=# \dm`
\dn[S+] [PATTERN]	Lists all schemas (namespace). If **+** is added to the command, the permission and description of each schema are listed.	If **PATTERN** is specified, only schemas whose names match the pattern are shown. By default, only schemas you created are displayed.	Lists information about all schemas whose names start with **d**. `openGauss=# \dn+ d*`
\do[S] [PATTERN]	Lists available operators with their operand and return types.	If **PATTERN** is specified, only operators whose names match **PATTERN** are displayed. By default, only operators you created are displayed.	Lists available operators with their operand and return types. `openGauss=# \do`
\dO[S+] [PATTERN]	Lists collations.	If **PATTERN** is specified, only rules whose names match **PATTERN** are displayed. By default, only collations you created are displayed.	Lists collations. `openGauss=# \dO`

Parameter	Description	Value Range	Example

\dp [PATTERN]	Lists tables, views, and related permissions.
	The following result about **\dp** is displayed:
	`rolename=xxxx/yyyy  -- Assigning permissions to a role`
	`=xxxx/yyyy  --Assigning permissions to public`
	xxxx indicates assigned permissions, and *yyyy* indicates roles with the assigned permissions. For details on permission descriptions, see **Table 2-19**.
		If **PATTERN** is specified, only tables and views whose names match the pattern are shown.	Lists tables, views, and related permissions.
			`openGauss=# \dp`

\drds [PATTERN1 [PATTERN2]]	Lists all modified configuration parameters. These settings can be for roles, for databases, or for both. **PATTERN1** and **PATTERN2** indicate a role pattern and a database pattern, respectively.	If **PATTERN** is specified, only rules whose names match **PATTERN** are displayed. If the default value is used or ***** is specified, all settings are listed.	Lists all modified configuration parameters of the **postgres** database. `openGauss=# \drds *openGausss`
\dT[S+] [PATTERN]	Lists all data types.	If **PATTERN** is specified, only types whose names match **PATTERN** are shown.	Lists all data types. `openGauss=# \dT`
\du[+] [PATTERN]	Lists all database roles. **NOTE** Since the concepts of "users" and "groups" have been unified into "roles", this command is equivalent to **\dg**. To be compatible with earlier versions, two commands are reserved.	If **PATTERN** is specified, only roles whose names match **PATTERN** are listed.	Lists all database roles. `openGauss=# \du`

Parameter	Description	Value Range	Example
\dE[S+] [PATTERN] \di[S+] [PATTERN] \ds[S+] [PATTERN] \dt[S+] [PATTERN] \dv[S+] [PATTERN]	In this group of commands, the letters E, i, s, t, and v stand for a foreign table, index, sequence, table, or view, respectively. You can specify any or a combination of these letters sequenced in any order to obtain an object list. For example, **\dit** lists all indexes and tables. If a command is suffixed with a plus sign (+), physical dimensions and related descriptions of each object will be displayed.	If **PATTERN** is specified, only objects whose names match **PATTERN** are displayed. By default, only objects you created are displayed. You can specify **PATTERN** or **S** to view other system objects.	Lists all indexes and views. `openGauss=# \div`
\dx[+] [PATTERN]	Lists installed extensions.	If **PATTERN** is specified, only extensions whose names match **PATTERN** are displayed.	Lists installed extensions. `openGauss=# \dx`
\l[+]	Lists the names, owners, character set encodings, and permissions of all the databases on the server.	-	Lists the names, owners, character set encodings, and permissions of all the databases on the server. `openGauss=# \l`

\sf[+] FUNCNAME	Shows function definitions.
	NOTE
	If the function name contains parentheses, enclose the function name with quotation marks and add the parameter type list following the double quotation marks. Also enclose the list with parentheses.
		-	Assume a function **function_a** and a function **func()name**. This parameter will be as follows:
			`openGauss=# \sf function_a`
			`openGauss=# \sf "func()name"(argtype1, argtype2)`

Parameter	Description	Value Range	Example
\z [PATTERN]	Lists all tables, views, and sequences in the database and their access permissions.	If a pattern is given, it is regarded as a regular expression, and only matched tables, views, and sequences are displayed.	Lists all tables, views, and sequences in the database and their access permissions. `openGauss=# \z`

Table 2-19 Permissions parameters

Parameter	Description
r	SELECT: allows users to read data from specified tables and views.
w	UPDATE: Allows users to update columns in specified tables.
a	INSERT: allows users to insert data to specified tables.
d	DELETE: allows users to delete data from specified tables.
D	TRUNCATE: allows users to delete all data from specified tables.
x	REFERENCES: allows users to create foreign key constraints.
t	TRIGGER: allows users to create a trigger on specified tables.
X	EXECUTE: allows users to use specified functions and the operators that are realized by the functions.
U	USAGE: ● For procedural languages, allows users to specify a procedural language when creating a function. ● For schemas, allows users to access objects includes in specified schemas. ● For sequences, allows users to use the nextval function.
C	CREATE: ● For databases, allows users to create schemas within a database. ● For schemas, allows users to create objects in a schema. ● For tablespaces, allows users to create tables in a tablespace and set the tablespace to default one when creating databases and schemas.
c	CONNECT: allows users to connect to specified databases.

Parameter	Description
T	TEMPORARY: allows users to create temporary tables.
A	ALTER: allows users to modify the attributes of specified objects.
P	DROP: allows users to delete a specified object.
m	COMMENT: allows users to define or modify comments of specified objects.
i	INDEX: allows users to create indexes on specified tables.
v	VACUUM: allows users to perform ANALYZE and VACUUM operations on specified tables.
*	Authorization options for preceding permissions

Table 2-20 Formatting meta-commands

Parameter	Description
\a	Controls the switchover between unaligned mode and aligned mode.
\C [STRING]	Sets the title of any table being printed as the result of a query or cancels such a setting.
\f [STRING]	Specifies a field separator for unaligned query output.
\H	● If the text format schema is used, switches to the HTML format schema. ● If the HTML format schema is used, switches to the text format.
\pset NAME [VALUE]	Sets options affecting the output of query result tables. For the value of NAME, see **Table 2-21**.
\t [on|off]	Switches the information and row count footer of the output column name.
\T [STRING]	Specifies attributes to be placed within the table tag in HTML output format. If the parameter is not configured, the attributes are not configured.
\x [on|off|auto]	Switches expanded table formatting mode.

Table 2-21 Adjustable printing options

Option	Description	Value Range
border	The value must be a number. In general, a larger number indicates wider borders and more table lines.	● The value is an integer greater than **0** in HTML format. ● The value range in other formats is as follows: – **0**: no border – **1**: internal dividing line – **2**: table frame
expanded (or x)	Switches between regular and expanded formats.	● When expanded format is enabled, query results are displayed in two columns, with the column name on the left and the data on the right. This format is useful if the data does not fit the screen in the normal "horizontal" format. ● The expanded format is used when the query output is wider than the screen. Otherwise, the regular format is used. The regular format is effective only in the aligned and wrapped formats.
fieldsep	Specifies the field separator to be used in unaligned output format. In this way, you can create tab- or comma-separated output required by other programs. To set a tab as field separator, type **\pset fieldsep '\t'**. The default field separator is a vertical bar (|).	-
fieldsep_zero	Specifies the field separator to be used in unaligned output format to zero bytes.	-
footer	Enables or disables the display of table footers.	-

Option	Description	Value Range
format	Selects the output format. Unique abbreviations are allowed. (This means a single letter is sufficient.)	Value range: ● **unaligned**: Write all columns of a row on one line, separated by the currently active column separator. ● **aligned**: This format is standard and human-readable. ● **wrapped**: This format is similar to **aligned**, but includes the packaging cross-line width data value to suit the width of the target field output. ● **html**: This format output table to the markup language for a document. The output is not a complete document. ● **latex**: This format output table to the markup language for a document. The output is not a complete document. ● **troff-ms:** This format output table to the markup language for a document. The output is not a complete document.
null	Specifies a string to be printed in place of a null value.	By default, nothing is printed, which can easily be mistaken for an empty character string.
numericlo cale	Enables or disables the display of a locale-specific character to separate groups of digits to the left of the decimal marker.	● **on**: The specified separator is displayed. ● **off**: The specified separator is not displayed If this parameter is ignored, the default separator is displayed.

Option	Description	Value Range
pager	Controls the use of a pager for query and gsql help outputs. If the PAGER environment variable is specified, the output is piped to the specified program. Otherwise, a platform-dependent default is used.	● **on**: The pager is used for terminal output that does not fit the screen. ● **off**: The pager is not used. ● **always**: The pager is used for all terminal output regardless of whether it fits the screen.
recordsep	Specifies the record separator to be used in unaligned output format.	-
recordsep_zero	Specifies the record separator to be used in unaligned output format to zero bytes.	-
tableattr (or T)	Specifies attributes to be placed inside the HTML table tag in HTML output format (such as cellpadding or bgcolor). Note that you do not need to specify border here because it has been used by **\pset border**. If no value is given, the table attributes do not need to be set.	-
title	Specifies the table title for any subsequently printed tables. This can be used to give your output descriptive tags. If no value is given, the title does not need to be set.	-
tuples_only (or t)	Enables or disables the tuples-only mode. Full display may show extra information, such as column headers, titles, and footers. In tuples_only mode, only the table data is displayed.	-
feedback	Specifies whether to output the number of result lines.	-

Table 2-22 Connection meta-commands

Parameter	Description	Value Range
\c[onnect] [DBNAME|- USER|- HOST|- PORT|-]	Connects to a new database. (The current database is **postgres**.) If a database name contains more than 63 bytes, only the first 63 bytes are valid and are used for connection. However, the database name displayed in the command line of **gsql** is still the name before the truncation. **NOTE** If the database login user is changed during reconnection, you need to enter the password of the new user. The maximum length of the password is 999 bytes, which is restricted by the maximum value of **password_max_length**.	-
\encoding [ENCODING]	Specifies the client character set encoding.	This command shows the current encoding if it has no parameter.
\conninfo	Outputs information about the current database connection.	-

Table 2-23 OS meta-commands

Parameter	Description	Value Range
\cd [DIR]	Changes the current working directory.	An absolute path or relative path that meets the OS path naming convention
\setenv NAME [VALUE]	Sets the **NAME** environment variable to **VALUE**. If **VALUE** is not provided, do not configure the environment variable.	-
\timing [on|off]	Toggles a display of how long each SQL statement takes, in milliseconds.	● **on**: specifies that the display is enabled. ● **off**: indicates that the display is disabled.
\! [COMMAND]	Escapes to a separate Unix shell or runs a Unix command.	-

Table 2-24 Variable meta-command

Parameter	Description
\prompt [TEXT] NAME	Prompts the user to specify the variable name in text format.
\set [NAME [VALUE]]	Sets the *NAME* internal variable to **VALUE**. If more than one value is provided, *NAME* is set to the concatenation of all of them. If only one parameter is provided, the variable is configured with an empty value. Some common variables are processed differently in **gsql** and they are combinations of uppercase letters, numbers and underscores. **Table 2-25** is a list of variables that are specially treated.
\unset NAME	Deletes the variable name of **gsql**.

Table 2-25 Common \set commands

Name	Description	Value Range
\set VERBOSITY value	This variable can be set to **default**, **verbose**, or **terse** to control redundant lines of error reports.	Value range: **default**, **verbose**, and **terse**
\set ON_ERROR_STOP value	If this variable is set, the script execution stops immediately. If this script is invoked from another script, that script execution will stop immediately as well. If the primary script is invoked using the **-f** option rather than from one **gsql** session, **gsql** will return error code 3, indicating the difference between the current error and critical errors. (The error code for critical errors is 1.)	Value range: **on/off**, **true/false**, **yes/no**, and **1/0**

Name	Description	Value Range
\set AUTOCOMMIT [on|off]	Sets the auto commit behavior of the current gsql connection. **on** indicates that auto commit is enabled, and **off** indicates that auto commit is disabled. By default, the gsql connection is automatically committed, and each individual statement is implicitly committed. If auto commit is disabled for performance or other purposes, you need to explicitly run the COMMIT command to ensure that transactions are committed. For example, execute the COMMIT statement to explicitly commit transactions after a specified service SQL statement is executed. Particularly, ensure that all transactions are committed before the gsql client exits. **NOTE** The auto commit is enabled in gsql by default. If you disable it, all the statements executed later will be packaged in implicit transactions, and you cannot execute statements that cannot be executed within transactions.	• **on**: The auto commit is enabled. • **off**: The auto commit is disabled.

Name	Description	Value Range
\set RETRY [retry_times]	Determines whether to enable the retry function if statement execution encounters errors. The parameter **retry_times** specifies the maximum number of retry times and the default value is **5**. Its value ranges from **5** to **10**. If the retry function has been enabled, when you run the **\set RETRY** command again, the retry function will be disabled. The configuration file **retry_errcodes.conf** shows a list of errors. If these errors occur, retry is required. This configuration file is placed in the same directory as that for executable **gsql** programs. This configuration file is configured by the system rather than by users and cannot be modified by the users. The retry function can be used in the following error scenarios: ● **YY002**: TCP communication errors. Print information: **Connection reset by peer**. (reset between DNs) ● YY003: Lock timeout. Print information: **Lock wait timeout.../wait transaction** *xxx* **sync time exceed** *xxx*. ● YY004: TCP communication error. Print information: **Connection timed out**. ● YY005: Failed to issue SET commands. Print information: **ERROR SET query**. ● YY006: Failed to apply for memory. Print information: **memory is temporarily unavailable**. ● YY007: Communication library error. Print information: **Memory allocate error**. ● YY008: Communication library error. Print information: **No data in buffer**. ● YY009: Communication library error. Print information: **Close because release memory**. ● **YY010**: Communication library error. Print information: **TCP disconnect**. ● **YY011**: Communication library error. Print information: **SCTP disconnect**. (Due to specification changes, the current version no longer supports the current feature. Do not use this feature.)	Value range of **retry_times**: **5** to **10**

Name	Description	Value Range
	● **YY012**: Communication library error. Print information: **Stream closed by remote**. ● **YY013**: Communication library error. Print information: **Wait poll unknown error**. If an error occurs, **gsql** queries connection status of all DNs. If the connection status is abnormal, **gsql** sleeps for 1 minute and tries again. In this case, the retries in most of the primary/standby switchover scenarios are involved. **NOTE** 1. Statements in transaction blocks cannot be retried upon a failure. 2. Retry is not supported if errors are found using ODBC or JDBC. 3. For SQL statements with unlogged tables, the retry is not supported if a node is faulty. 4. For **gsql** client faults, the retry is not supported.	

Table 2-26 Large object meta-commands

Parameter	Description
\lo_list	Displays a list of all GaussDB large objects stored in the database, along with the comments provided for them.

PATTERN

The various **\d** commands accept a **PATTERN** parameter to specify the object name to be displayed. In the simplest case, a pattern is just the exact name of the object. The characters within a pattern are normally folded to lower case, similar to those in SQL names. For example, **\dt FOO** will display the table named **foo**. As in SQL names, placing double quotation marks (") around a pattern prevents them being folded to lower case. If you need to include a double quotation mark (") in a pattern, write it as a pair of double quotation marks ("") within a double-quote sequence, which is in accordance with the rules for SQL quoted identifiers. For example, **\dt "FOO""BAR"** will be displayed as a table named **FOO"BAR** instead of **foo"bar**. You cannot put double quotation marks around just part of **PATTERN**, which is different from the normal rules for SQL names. For example, **\dt FOO"FOO"BAR** will be displayed as a table named **fooFOObar** if just part of a pattern is quoted.

Whenever the **PATTERN** parameter is omitted completely, the **\d** commands display all objects that are visible in the current schema search path, which is equivalent to using an asterisk (*) as the pattern. An object is regarded to be visible if it can be referenced by name without explicit schema qualification. To see all objects in the database regardless of their visibility, use a dot within double asterisks (*.*) as the **PATTERN**.

Within a pattern, the asterisk (*) matches any sequence of characters (including no characters) and a question mark (?) matches any single character. This notation is comparable to Unix shell file name patterns. For example, **\dt int*** displays tables whose names begin with **int**. But within double quotation marks, the asterisk (*) and the question mark (?) lose these special meanings and are just matched literally.

A pattern that contains a dot (.) is interpreted as a schema name pattern followed by an object name pattern. For example, **\dt foo*.*bar*** displays all tables (whose names include **bar**) in schemas starting with **foo**. If no dot appears, then the pattern matches only visible objects in the current schema search path. Again, a dot within double quotation marks loses its special meaning and is matched literally.

Advanced users can use regular-expression notations, such as character classes. For example, **[0-9]** can be used to match any digit. All regular expression special characters work as POSIX regular expressions specify. The following characters are not allowed:

- A dot (.) is used as a separator.
- An asterisk (*) is translated into an asterisk prefixed with a dot (.*), which is a regular-expression marking.
- A question mark (?) is translated into a dot (.).
- A dollar sign ($) is matched literally.

You can write ?, (*R*+|), (*R*|), and *R* to the pattern characters (., *R**, and *R*?). The dollar sign ($) does not need to work as a regular-expression character since the pattern must match the whole name, which is different from the usual interpretation of regular expressions. In other words, the dollar sign ($) is automatically appended to your pattern. If you do not expect a pattern to be anchored, write an asterisk (*) at its beginning or end. All regular-expression special characters within double quotation marks lose their special meanings and are matched literally. Regular-expression special characters in operator name patterns (such as the **\do** parameter) are also matched literally.

2.9.6 Troubleshooting

Low Connection Performance

- **log_hostname** is enabled, but DNS is incorrect.

  Connect to the database, run **show log_hostname** to check whether **log_hostname** is enabled in the database.

  If it is enabled, the database will use DNS to check the name of the host where the client is deployed. If the database is configured with an incorrect or unreachable DNS server, the database connection will take a long time to set up. For details about this parameter, see the description of **log_hostname** in section "GUC Parameters" > "Error Reporting and Logging" > "Logging Content".

Problems in Setting Up Connections

- gsql: could not connect to server: No route to host

  This problem occurs generally because an unreachable IP address or port number was specified. Check whether the values of **-h** and **-p** parameters are correct.

- gsql: FATAL: Invalid username/password,login denied.

  This problem occurs generally because an incorrect username or password was entered. Contact the database administrator to check whether the username or password is correct.

- After **-h 127.0.0.1** is added to a DN, the DN can be connected to the database. After **-h 127.0.0.1** is deleted, the DN cannot be connected to the database.

  Run the **show unix_socket_directory** SQL statement to check whether the Unix socket directory used by the DN is the same as that specified by **$PGHOST** in the **shell** directory.

  If they are different, set **$PGHOST** to the directory specified by **unix_socket_directory**.

  For details about **unix_socket_directory**, see section "GUC Parameters" > "Connection and Authentication" > "Connection Settings".

- The "libpq.so" loaded mismatch the version of gsql, please check it.

  This problem occurs because the version of **libpq.so** used in the environment does not match that of **gsql**. Run the **ldd gsql** command to check the version of the loaded **libpq.so**, and then load correct **libpq.so** by modifying the environment variable **LD_LIBRARY_PATH**.

- gsql: symbol lookup error: xxx/gsql: undefined symbol: libpqVersionString

  This problem occurs because the version of **libpq.so** used in the environment does not match that of **gsql** (or the PostgreSQL **libpq.so** exists in the environment). Run the **ldd gsql** command to check the version of the loaded **libpq.so**, and then load correct **libpq.so** by modifying the environment variable **LD_LIBRARY_PATH**.

- gsql: connect to server failed: Connection timed out

  Is the server running on host "xx.xxx.xxx.xxx" and accepting TCP/IP connections on port xxxx?

  This problem is caused by network connection faults. Check the network connection between the client and the database server. If you cannot ping from the client to the database server, the network connection is abnormal. Contact network management personnel for troubleshooting.

  ```
  ping -c 4 10.10.10.1
  PING 10.10.10.1 (10.10.10.1) 56(84) bytes of data.
  From 10.10.10.1: icmp_seq=2 Destination Host Unreachable
  From 10.10.10.1 icmp_seq=2 Destination Host Unreachable
  From 10.10.10.1 icmp_seq=3 Destination Host Unreachable
  From 10.10.10.1 icmp_seq=4 Destination Host Unreachable
  --- 10.10.10.1 ping statistics ---
  4 packets transmitted, 0 received, +4 errors, 100% packet loss, time 2999ms
  ```

- gsql: FATAL: sorry, too many clients already, active/non-active: 197/3.

  This problem occurs because the number of system connections exceeds the allowed maximum. Contact the database administrator to release unnecessary sessions.

You can check the number of connections as described in **Table 2-27**.

You can view the session status in the **PG_STAT_ACTIVITY** view. To release unnecessary sessions, use the **pg_terminate_backend** function.

```
select datid,pid,state from pg_stat_activity;
 datid |       pid       | state
-------+-----------------+--------
 13205 | 139834762094352 | active
 13205 | 139834759993104 | idle
(2 rows)
```

The **pid** value is the thread ID of the session. Terminate the session using its thread ID.

```
SELECT PG_TERMINATE_BACKEND(139834759993104);
```

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

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

Table 2-27 Viewing the number of session connections

Description	Command

View the maximum number of sessions connected to a specific user.	Run the following command to view the upper limit of user **user1**'s connections. **-1** indicates that no connection upper limit is set for user **user1**.
	`SELECT ROLNAME,ROLCONNLIMIT FROM PG_ROLES WHERE ROLNAME='user1';`
	` rolname | rolconnlimit`
	`---------+--------------`
	` user1    |           -1`
	`(1 row)`

View the number of session connections that have been used by a user.	Run the following command to view the number of connections that have been used by **user1**. **1** indicates the number of connections that have been used by **user1**.
	`SELECT COUNT(*) FROM dv_sessions WHERE USERNAME='user1';`
	
	` count`
	`-------`
	`     1`
	`(1 row)`

View the maximum number of sessions connected to a specific database.	Run the following commands to view the upper limit of the number of **postgres**'s session connections. **–1** indicates that no upper limit is set for the number of **postgres**'s session connections.
	`SELECT DATNAME,DATCONNLIMIT FROM PG_DATABASE WHERE DATNAME='postgres';`
	
	` datname  | datconnlimit`
	`----------+--------------`
	` postgres |           -1`
	`(1 row)`

Description	Command

View the number of session connections that have been used by a specific database.	Run the following commands to view the number of session connections that have been used by **postgres**. **1** indicates the number of session connections that have been used by **postgres**.
	`SELECT COUNT(*) FROM PG_STAT_ACTIVITY WHERE DATNAME='postgres';`
	` count`
	`-------`
	`     1`
	`(1 row)`

View the number of session connections that have been used by all users.	Run the following commands to view the number of session connections that have been used by all users:
	`SELECT COUNT(*) FROM dv_sessions;`
	
	` count`
	`-------`
	`    10`
	`(1 row)`

- gsql: wait xxx.xxx.xxx.xxx:xxxx timeout expired

  When **gsql** initiates a connection request to the database, a 5-minute timeout period is used. If the database cannot correctly authenticate the client request and client identity within this period, **gsql** will exit the connection process for the current session, and will report the above error.

  Generally, this problem is caused by the incorrect host and port (that is, the *xxx* part in the error information) specified by the **-h** and **-p** parameters. As a result, the communication fails. Occasionally, this problem is caused by network faults. To resolve this problem, check whether the host name and port number of the database are correct.

- gsql: could not receive data from server: Connection reset by peer.

  Check whether DN logs contain information similar to " FATAL: cipher file "/data/coordinator/server.key.cipher" has group or world access". This error is usually caused by tampering with the permissions for data directories or some key files by mistake. For details about how to correct the permissions, see related permissions for files on other normal instances.

Other Faults

- There is a core dump or abnormal exit due to the bus error.

  Generally, this problem is caused by changes in loading the shared dynamic library (.so file in Linux) during process running. Alternatively, if the process binary file changes, the execution code for the OS to load machines or the entry for loading a dependent library will change accordingly. In this case, the OS kills the process for protection purposes, generating a core dump file.

  To resolve this problem, try again. In addition, do not run service programs in a database during O&M operations, such as an upgrade, preventing such a problem caused by file replacement during the upgrade.

NOTE

A possible stack of the core dump file contains dl_main and its function calling. The file is used by the OS to initialize a process and load the shared dynamic library. If the process has been initialized but the shared dynamic library has not been loaded, the process cannot be considered completely started.

3 Development and Design Proposal

3.1 Development and Design Proposal

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

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

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

3.2 Database Object Naming Conventions

The name of a database object must meet the following requirements: The name of a non-time series table ranges from 1 to 63 bytes and that of a time series table ranges from 1 to 53 characters. The name must start with a letter or underscore (_), and can contain letters, digits, underscores (_), dollar signs ($), and number signs (#).

- [Proposal] Do not use reserved or non-reserved keywords to name database objects.

  NOTE

  You can use the select * from pg_get_keywords() query GaussDB keyword or view the keyword in **Keywords**.
- [Proposal] Do not use a string enclosed in double quotation marks ("") to define the database object name, unless you need to specify its capitalization. Case sensitivity of database object names makes problem location difficult.
- [Proposal] Use the same naming format for database objects.
  - In a system undergoing incremental development or service migration, you are advised to comply with its historical naming conventions.

  - You are advised to use multiple words separated with underscores (_).
  - You are advised to use intelligible names and common acronyms or abbreviations for database objects. Acronyms or abbreviations that are generally understood are recommended. For example, you can use English words or Chinese pinyin indicating actual business terms. The naming format should be consistent within a database instance.
  - A variable name must be descriptive and meaningful. It must have a prefix indicating its type.
- [Proposal] The name of a table object should indicate its main characteristics, for example, whether it is an ordinary, temporary, or unlogged table.
  - An ordinary table name should indicate the business relevant to a dataset.
  - Temporary tables are named in the format of **tmp_***Suffix*.
  - Unlogged tables are named in the format of **ul_***Suffix*.
  - Foreign tables are named in the format of **f_***Suffix*.
  - Do not create database objects whose names start with **redis_**.
  - Do not create database objects whose names start with **mlog_** or **matviewmap_**.
- [Proposal] The name of a non-time series table object shall not exceed 63 bytes. If the length exceeds this value, the kernel truncates the table name. As a result, the table name is inconsistent with the configured name. In different character sets, characters may be truncated and unexpected characters may appear.

3.3 Database Object Design

3.3.1 Database and Schema Design

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

- You are advised to use schemas to isolate services for convenience and resource sharing.
- It is recommended that system administrators create schemas and databases and then assign required permissions to users.

Database Design

- [Rule] Create databases as required by your service. Do not use the default **postgres** database of a database instance.
- [Proposal] Create a maximum of three user-defined databases in a database instance.
- [Proposal] To make your database compatible with most characters, you are advised to use the UTF-8 encoding when creating a database.

- [Notice] When you create a database, exercise caution when you set **ENCODING** and **DBCOMPATIBILITY** configuration items. GaussDB supports the A, B, C, and PG compatibility modes, which are compatible with the Oracle syntax, MySQL syntax, Teradata syntax, and Postgres syntax, respectively. The syntax behavior varies according to the compatibility mode. By default, the A compatibility mode is used.
- [Notice] By default, a database owner has all permissions for all objects in the database, including the deletion permission. Exercise caution when deleting a permission.

Schema Design

- [Notice] To let a user access an object in a schema, assign the usage permission and the permissions for the object to the user, unless the user has the **sysadmin** permission or is the schema owner.
- [Notice] To let a user create an object in the schema, grant the create permission for the schema to the user.
- [Notice] By default, a schema owner has all permissions for all objects in the schema, including the deletion permission. Exercise caution when deleting a permission.

3.3.2 Table Design

Comply with the following principles to properly design a table:

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

Selecting a Storage Mode

[Proposal] Selecting a storage model is the first step in defining a table. The storage model mainly depends on the customer's service type. For details, see **Table 3-1**.

Table 3-1 Table storage modes and scenarios

Storage Type	Application Scenario
Row store	• Point queries (simple index-based queries that only return a few records). • Scenarios requiring frequent addition, deletion, and modification.
Column store	• Statistical analysis queries (requiring a large number of association and grouping operations). • Ad hoc queries (using uncertain query conditions and unable to utilize indexes to scan row-store tables).

Selecting a Partitioning Mode

If a table contains a large amount of data, partition the table based on the following rules:

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

Table 3-2 Table partitioning modes and scenarios

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

The example of a partitioned table definition is as follows:

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

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

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

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

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

3.3.3 Field Design

Selecting a Data Type

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

- [Proposal] Use the most efficient data types allowed.

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

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

Common String Types

Every column requires a data type suitable for its data characteristics. The following table lists common string types in GaussDB.

Table 3-3 Common string types

Name	Description	Max. Storage Capacity
CHAR(n)	Fixed-length string, where *n* indicates the stored bytes. If the length of an input string is smaller than *n*, the string is automatically padded to *n* bytes using null characters.	10 MB
CHARACTER(n)	Fixed-length string, where *n* indicates the stored bytes. If the length of an input string is smaller than *n*, the string is automatically padded to *n* bytes using null characters.	10 MB
NCHAR(n)	Fixed-length string, where *n* indicates the stored bytes. If the length of an input string is smaller than *n*, the string is automatically padded to *n* bytes using null characters.	10 MB
BPCHAR(n)	Fixed-length string, where *n* indicates the stored bytes. If the length of an input string is smaller than *n*, the string is automatically padded to *n* bytes using null characters.	10 MB
VARCHAR(n)	Variable-length string, where *n* indicates the maximum number of bytes that can be stored.	10 MB
CHARACTER VARYING(n)	Variable-length string, where *n* indicates the maximum number of bytes that can be stored. This data type and VARCHAR(n) are different representations of the same data type.	10 MB
VARCHAR2(n)	Variable-length string, where *n* indicates the maximum number of bytes that can be stored. This data type is added to be compatible with the Oracle database, and its behavior is the same as that of VARCHAR(n).	10 MB
NVARCHAR2(n)	Variable-length string, where *n* indicates the maximum number of bytes that can be stored.	10 MB

Name	Description	Max. Storage Capacity
TEXT	Variable-length string. Its maximum length is 1 GB minus 8203 bytes.	1 GB minus 8203 bytes

3.3.4 Constraint Design

DEFAULT and NULL Constraints

- [Proposal] If all the column values can be obtained from services, you are not advised to use the **DEFAULT** constraint. Otherwise, unexpected results will be generated during data loading.
- [Proposal] Add **NOT NULL** constraints to columns that never have NULL values. The optimizer automatically optimizes the columns in certain scenarios.
- [Proposal] Explicitly name all constraints excluding **NOT NULL** and **DEFAULT**.

Partial Cluster Keys

A partial cluster key (PCK) is a local clustering technology used for column-store tables. After creating a PCK, you can quickly filter and scan fact tables using min or max sparse indexes in GaussDB. Comply with the following rules to create a PCK:

- [Notice] Only one PCK can be created in a table. A PCK can contain multiple columns, preferably no more than two columns.
- [Proposal] Create a PCK on simple expression filter conditions in a query. Such filter conditions are usually in the form of **col op const**, where **col** specifies a column name, **op** specifies an operator (such as =, >, >=, <=, and <), and **const** specifies a constant.
- [Proposal] If the preceding conditions are met, create a PCK on the column having the most distinct values.

Unique Constraints

- [Notice] Both row-store and column-store tables support unique constraints.
- [Proposal] The constraint name should indicate that it is a unique constraint, for example, **UNI***Included columns*.

Primary Key Constraints

- [Notice] Both row-store and column-store tables support primary key constraints.
- [Proposal] The constraint name should indicate that it is a primary key constraint, for example, **PK***Included columns*.

Check Constraints

- [Notice] Check constraints can be used in row-store tables but not in column-store tables.
- [Proposal] The constraint name should indicate that it is a check constraint, for example, **CK***Included columns*.

3.3.5 View and Joined Table Design

View Design

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

Joined Table Design

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

3.4 Tool Interconnection

3.4.1 JDBC Configuration

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

Connection Parameters

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

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

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

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

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

[Proposal] In precision-sensitive scenarios, the numeric type is recommended.

- [Proposal] When connecting to the database through JDBC, ensure that the following three time zones are the same:
  - Time zone of the host where the JDBC client is located
  - Time zone of the host where the GaussDB instance is located.
  - Time zone used during GaussDB instance configuration.

  NOTE

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

fetchsize

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

autocommit

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

Connection Releasing

[Proposal] You are advised to use connection pools to limit the number of connections from applications. Do not connect to a database every time you run an SQL statement.

[Proposal] After an application completes its tasks, disconnect its connection to GaussDB to release occupied resources. You are advised to set the session timeout interval in the jobs.

[Proposal] Reset the session environment before releasing connections to the JDBC connection tool. Otherwise, historical session information may cause object conflicts.

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

CopyManager

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

3.5 SQL Compilation

DDL

- [Proposal] In GaussDB, you are advised to execute DDL operations, such as creating table or making comments, separately from batch processing jobs to avoid performance deterioration caused by many concurrent transactions.
- [Proposal] Execute data truncation after unlogged tables are used because GaussDB cannot ensure the security of unlogged tables in abnormal scenarios.
- [Proposal] Suggestions on the storage mode of temporary and unlogged tables are the same as those on base tables. Create temporary tables in the same storage mode as the base tables to avoid high computing costs caused by hybrid row and column correlation.
- [Proposal] The total length of an index column cannot exceed 50 bytes. Otherwise, the index size will increase greatly, resulting in large storage cost and low index performance.
- [Proposal] Do not delete objects using **DROP...CASCADE**, unless the dependency between objects is specified. Otherwise, the objects may be deleted by mistake.

Data Loading and Unloading

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

Type Conversion

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

Query Operation

- [Proposal] Do not return a large number of result sets to a client except the ETL program. If a large result set is returned, consider modifying your service design.
- [Proposal] Perform DDL and DML operations encapsulated in transactions. Operations like table truncation, update, deletion, and dropping, cannot be

rolled back once committed. You are advised to encapsulate such operations in transactions so that you can roll back the operations if necessary.

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

Table 3-4 Time-related macros

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

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

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

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

  Convert the statement to the following:

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

- [Proposal] If an **in(val1, val2, val3...)** expression contains a large number of columns, you are advised to replace it with the **in (values (va11), (val2), (val3)...)** statement. The optimizer will automatically convert the **IN** constraint into a non-correlated subquery to improve the query performance.
- [Proposal] Replace **(not) in** with **(not) exist** when associated columns do not contain **NULL** values. For example, in the following query statement, if the **T1.C1** column does not contain any **NULL** value, add the **NOT NULL** constraint to the **T1.C1** column, and then rewrite the statements.

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

  Rewrite the statement as follows:

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

  NOTE

  - If the value of the T1.C1 column is not **NOT NULL**, the preceding rewriting cannot be performed.
  - If the **T1.C1** column is the output of a subquery, check whether the output is **NOT NULL** based on the service logic.

- [Proposal] Use cursors instead of the **LIMIT OFFSET** syntax to perform pagination queries to avoid resource overheads caused by multiple executions. A cursor must be used in a transaction, and you must disable the cursor and commit the transaction once the query is finished.

4 Best Practices

4.1 Best Practices of Table Design

4.1.1 Selecting a Storage Model

During database design, some key factors about table design will greatly affect the subsequent query performance of the database. Table design affects data storage as well. A good table design reduces I/O operations and minimizes memory usage, improving the query performance.

Selecting a model for table storage is the first step of table definition. Select a proper storage model for your service based on the following table:

Storage Model	Applicable Scenario
Row-store	Point queries (simple index-based queries that only return a few records) Scenarios requiring frequent addition, deletion, and modification operations
Column-store	Statistics analysis query, in which operations, such as group and join, are performed many times.

4.1.2 Using Partial Cluster Keys (PCKs)

PCK is a column-store technology. It can minimize or maximize sparse indexes to quickly filter base tables. PCKs can be used to specify multiple columns. However, it is recommended that a maximum of two columns be specified. Use the following principles to specify PCKs:

1. The specified PCKs must be restricted by simple expressions in base tables. Such constraints are usually represented by *col op const*, in which *col* indicates the column name, *op* indicates operators (including =, >, >=, <=, and <), and *const* indicates constants.

2. Select columns that are frequently selected (to filter much more undesired data) in simple expressions.
3. List the most frequently selected columns at the top.
4. List the columns of the enumerated type at the top.

4.1.3 Using Partitioned Tables

A partitioned table is a logical table that is divided into several physical partitions based on a specific plan. The table based on the logic is called a partitioned table, and each physical block is called a partition. A partitioned table is a logical table and does not store data. Data is actually stored in partitions. A partitioned table has the following advantages over an ordinary table:

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

   Partitioned tables supported by the GaussDB database are level-1 and level-2 partitioned tables. Level-1 partitioned tables include range partitioned tables, interval partitioned tables, list partitioned tables, and hash partitioned tables. Level-2 partitioned tables include nine combinations of any two of range partitioned tables, list partitioned tables, and hash partitioned tables.

   – Range partitioned table: Data within a certain range is mapped to each partition. The range is determined by the partition key specified when the partitioned table is created. This partitioning method is most commonly used. The partition key is usually a date. For example, sales data is partitioned by month.
   – Interval partitioned table: a special type of range partitioned tables. Compared with range partitioned tables, interval value definition is added. When no matching partition can be found for an inserted record, a partition can be automatically created based on the interval value.
   – List partitioned table: Key values contained in the data are stored in different partitions, and the data is mapped to each partition in sequence. The key values contained in the partitions are specified when the partitioned table is created.
   – Hash partitioned table: Data is mapped to each partition based on the internal hash algorithm. The number of partitions is specified when the partitioned table is created.
   – Level-2 partitioned table: a partitioned table obtained by randomly combining range partitioning, list partitioning, and hash partitioning. Both level-1 and level-2 partitions can be defined in the preceding three ways.

4.1.4 Selecting a Data type

Efficient data types include the following:

1. **Select data types that facilitate data calculation.**

Generally, the calculation of integers (including common comparison calculations, such as =, >, <, ≥, ≤, and ≠ and **GROUP BY**) is more efficient than that of strings and floating point numbers. For example, if you need to perform a point query on a column-store table whose numeric column is used as a filter criterion, the query will take over 10s. If you change the data type from **NUMERIC** to **INT**, the query will be reduced to about 1.8s.

2. **Select data types with a short length.**

   Data types with a short length reduce both the data file size and the memory used for computing, improving the I/O and computing performance. For example, use **SMALLINT** instead of **INT**, and **INT** instead of **BIGINT**.

3. **Use the same data type for a join.**

   You are advised to use the same data type for a join. To join columns with different data types, the database needs to convert them to the same type, which leads to additional performance overheads.

4.2 Best Practices of Data Import

Using COPY to Import Data

The **COPY** statement imports data from local and remote databases in parallel. **COPY** imports large amounts of data more efficiently than using **INSERT** statements.

For details about how to use the **COPY** statement, see **Running the COPY FROM STDIN Statement to Import Data**.

Using a gsql Meta-Command to Import Data

The **\copy** command can be used to import data after you log in to a database through any **psql** client. Different from the **COPY** statement, the **\copy** command reads data from or writes to a file.

Data read or written using the **COPY** statement is transferred through the connection between the server and the client and may not be efficient. The **COPY** statement is recommended when the amount of data is large.

For details about how to use the **\copy** command, see **Using a gsql Meta-Command to Import Data**.

NOTE

\copy applies only to small-scale data import in good format. It does not preprocess invalid characters or provide error tolerance. Therefore, **\copy** cannot be used in scenarios where abnormal data exists. **COPY** is preferred for data import.

Using INSERT to Insert Multiple Rows

If the **\copy** command cannot be used for insertion and you need to insert SQL statements, you can use the **INSERT** statement for multiple rows as required. If you use a column-store table and insert one or more rows at a time, the data compression efficiency is low.

Multi-row inserts improve performance by bulk inserts. The following example inserts three rows into a three-column table using a single **INSERT** statement. This is still a small insert, shown simply to illustrate the syntax of a multi-row insert. For details about how to create a table, see **Creating and Managing Tables**.

To insert multiple rows of data to the table **customer_t1**, run the following command:

```
openGauss=# insert into customer_t1 values
(68, 'a1', 'zhou','wang'),
(43, 'b1', 'wu', 'zhao'),
(95, 'c1', 'zheng', 'qian');
```

For more details and examples, see **INSERT**.

Using INSERT for Bulk Insert

Use a bulk insert operation with a **SELECT** clause for high-performance data insertion.

Use the **INSERT** and **CREATE TABLE AS** statements when you need to move data or a subset of data from one table into another.

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

```
openGauss=# CREATE TABLE customer_t2
(
    c_customer_sk             integer,
    c_customer_id             char(5),
    c_first_name              char(6),
    c_last_name               char(8)
);
openGauss=# INSERT INTO customer_t2 SELECT * FROM customer_t1;
```

The preceding example is equivalent to:

```
openGauss=# CREATE TABLE customer_t2 AS SELECT * FROM customer_t1;
```

4.3 Best Practices of SQL Queries

Based on the SQL execution mechanism and a large number of practices, SQL statements can be optimized by following certain rules to enable the database to execute SQL statements more quickly and obtain correct results.

- Replace **UNION** with **UNION ALL**.

  UNION eliminates duplicate rows while merging two result sets but **UNION ALL** merges the two result sets without deduplication. Therefore, replace **UNION** with **UNION ALL** if you are sure that the two result sets do not contain duplicate rows based on the service logic.
- Add **NOT NULL** to the join columns.

  If there are many **NULL** values in the **JOIN** columns, you can add the filter criterion **IS NOT NULL** to filter data in advance to improve the **JOIN** efficiency.
- Replace **NOT IN** with **NOT EXISTS**.

  Nested Loop Anti Join must be used to implement **NOT IN**, and **Hash Anti Join** is required for **NOT EXISTS**. If no **NULL** value exists in the **JOIN** columns, **NOT IN** is equivalent to **NOT EXISTS**. Therefore, if you are sure that no **NULL** value exists, you can convert **NOT IN** to **NOT EXISTS** to generate **hash join** and to improve the query performance.

As shown in the following statement, the **t2.d2** column does not contain **NULL** values (it is set to **NOT NULL**) and **NOT EXISTS** is used for the query.

```
select * from t1 where not exists(select * from t2 where t1.c1 = t2.c1);
```

The generated execution plan is as follows:

```
openGauss=# explain select * from t1 where not exists(select * from t2 where t1.c1 = t2.c1);
QUERY PLAN
------------------------------------------------------------------
Hash Anti Join  (cost=58.35..107.44 rows=1074 width=8)
Hash Cond: (t1.c1 = t2.c1)
->  Seq Scan on t1  (cost=0.00..31.49 rows=2149 width=8)
->  Hash  (cost=31.49..31.49 rows=2149 width=4)
->  Seq Scan on t2  (cost=0.00..31.49 rows=2149 width=4)
(5 rows)
```

- Use **hashagg**.

  If a plan involving **groupAgg** and **SORT** operations generated by the **GROUP BY** statement is poor in performance, you can set **work_mem** to a larger value to generate a **hashagg** plan, which does not require sorting and improves the performance.

- Replace functions with **CASE** statements.

  The GaussDB performance greatly deteriorates if a large number of functions are called. In this case, you can change the pushdown functions to CASE statements.

- Do not use functions or expressions for indexes.

  Using functions or expressions for indexes will stop indexing and enable scanning on the full table.

- Do not use operators **!=**, **<**, **>** , **NULL**, **OR**, or implicit parameter conversion in **WHERE** clauses.

- Split complex SQL statements.

  You can split an SQL statement into several ones and save the execution result to a temporary table if the SQL statement is too complex to be tuned using the solutions above, including but not limited to the following scenarios:

  – The same subquery is involved in multiple SQL statements of a job and the subquery contains a large amount of data.

  – Incorrect **Plan cost** causes a small hash bucket of subquery. For example, the actual number of rows is 10 million, but only 1000 rows are in hash bucket.

  – Functions such as **substr** and **to_number** cause incorrect measures for subqueries containing a large amount of data.

  – **BROADCAST** subqueries are performed on large tables in multi-DN environment.

For details about optimization, see **Typical SQL Optimization Methods**.

5 Application Development Guide

5.1 Development Specifications

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

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

If you do not do so, the connection in the connection pool will be stateful, which affects subsequent operations on the connection pool.

Compatibility:

- The new driver is forward compatible with the database. To use the new features added to the driver and database, you must upgrade the database.
- Setting **behavior_compat_options** to **'proc_outparam_override'** is applicable only in A-compatible mode.

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

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

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

Table 5-1 Description of compatibility

Driver	Compatibility
JDBC and Go	The driver is forward compatible with earlier database versions. However, to use the new features added to the driver and database, you must upgrade the database.

Driver	Compatibility
ODBC, libpq, and Psycopg	The driver version must match the database version.

5.2 Obtaining the Driver Package

Obtaining the Driver Package

Download the required packages listed in **Table 5-2**.

Table 5-2 Driver package download list

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

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

1. Upload the software package and verification package to the same directory on a Linux VM.
2. Run the following command to verify the integrity of the software package:
   cat *GaussDB_driver.zip.sha256* **| sha256sum --check**
   If **OK** is displayed in the command output, the verification is successful.
   ```
   GaussDB_driver.zip: OK
   ```

5.3 Development Based on JDBC

Java Database Connectivity (JDBC) is a Java API for running SQL statements. It provides unified access APIs for different relational databases, based on which applications process data. The GaussDB library supports JDBC 4.0 and requires JDK 1.8 for code compiling. It does not support JDBC-ODBC bridge.

5.3.1 JDBC Package, Driver Class, and Environment Class

JDBC Package

Obtain **release package** provided by GaussDB.

Obtain the **GaussDB-Kernel_VxxxRxxxCxx.x-***OS version number***-64bit-Jdbc.tar.gz** package from the release package. After the decompression, you will obtain the following JDBC packages in .jar format:

- **gsjdbc4.jar**: The driver class name and loading path are the same as those of PostgreSQL driver, which facilitates the migration of services running on PostgreSQL. However, some interfaces supported by **gsjdbc4.jar** are different from those supported by PostgreSQL and need to be adjusted on the service side.
- **gsjdbc200.jar**: The driver class name and loading path are the same as those of GaussDB 200, which facilitates the migration of services running on GaussDB 200. However, some interfaces supported by **gsjdbc200.jar** are different from those supported by GaussDB 200 and need to be adjusted on the service side.
- **opengaussjdbc.jar**: The main class name is **com.huawei.opengauss.jdbc.Driver**. The URL prefix of the database connection is **jdbc:opengauss**. This driver package is recommended. This driver package is used when both PostgreSQL and GaussDB are accessed in a JVM process.

NOTICE

- The loading paths of driver classes in different driver packages are different, but the interface functions are the same.
- The **gsjdbc4** driver package cannot be used to operate the PostgreSQL database. Although the connection can be successfully established in some versions, some interface behaviors are different from those of PostgreSQL JDBC, which may cause unknown errors.
- The PostgreSQL driver package cannot be used to operate the GaussDB database. Although the connection can be successfully established in some versions, some interface behaviors are different from those of GaussDB JDBC, which may cause unknown errors.

Driver Class

Before establishing a database connection, load the **org.postgresql.Driver** database driver class.

NOTE

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

Environment Class

JDK 1.8 must be configured on the client. The configuration method is as follows:

Step 1 In the MS-DOS window, run **java -version** to check the JDK version. Ensure that the version is JDK 1.8. If JDK is not installed, download the installation package from the official website and install it.

Step 2 Configure system environment variables.

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

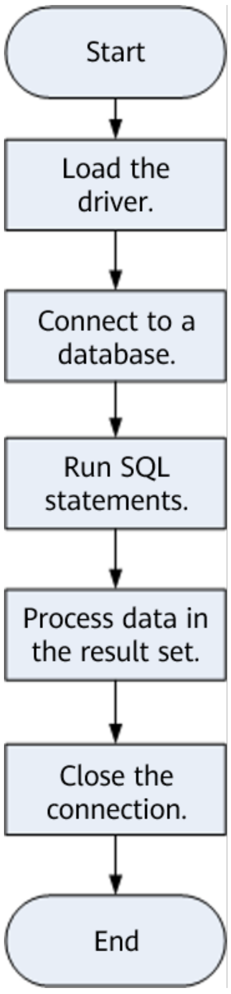
Table 5-3 Description

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

----End

5.3.2 Development Process

Figure 5-1 Application development process based on JDBC



5.3.3 Loading the Driver

Load the database driver before creating a database connection.

You can load the driver in the following ways:

- Before creating a connection, implicitly load the driver in the code: **Class.forName("org.postgresql.Driver")**
- During the JVM startup, transfer the driver as a parameter to the JVM: **java -Djdbc.drivers=org.postgresql.Driver jdbctest**

NOTE

jdbctest is the name of a test application.

5.3.4 Connecting to a Database

After a database is connected, you can use JDBC to run SQL statements to operate data.

Function Prototype

JDBC provides the following three database connection methods:

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

Parameters

Table 5-4 Database connection parameters

Parameter	Description
url	**postgresql.jar** database connection descriptor. The format is as follows: • jdbc:postgresql:database • jdbc:postgresql://*host*/*database* • jdbc:postgresql://*host*:*port*/*database* • jdbc:postgresql://*host*:*port*/*database*?*param1*=*value1*&*param2*=*value2* • jdbc:postgresql://*host1*:*port1*,*host2*:*port2*/*database*?*param1*=*value1*&*param2*=*value2* **NOTE** • **database** indicates the name of the database to connect. • **host** indicates the name or IP address of the database server. For security purposes, the primary database node forbids access from other nodes in the database without authentication. To access the primary database node from inside the database, deploy the JDBC program on the host where the primary database node is located and set **host** to **127.0.0.1**. Otherwise, the error message "FATAL: Forbid remote connection with trust method!" may be displayed. It is recommended that the service system be deployed outside the database. If it is deployed inside, database performance may be affected. By default, the local host is used to connect to the server. • **port** indicates the port number of the database server. By default, the database on port 5432 of the local host is connected. • **param** indicates a database connection attribute. The parameter can be configured in the URL. The URL starts with a question mark (?), uses an equal sign (=) to assign a value to the parameter, and uses an ampersand (&) to separate parameters. You can also use the attributes of the **info** object for configuration. For details, see the example below. • **value** indicates the database connection attribute values. • The **connectTimeout** and **socketTimeout** parameters must be set for connection. If they are not set, the default value **0** is used, indicating that the connection will not time out. When the network between the DN and client is faulty, the client does not receive the ACK packet from the DN. In this case, the client starts the timeout retransmission mechanism to continuously retransmit packets. A timeout error is reported only when the timeout interval reaches the default value **600s**. As a result, the RTO is high. • In a centralized environment, you are advised to configure all DNs for the connection strings and set the **targetServerType** parameter.

Parameter	Description
info	Database connection attributes (all attributes are case sensitive). Common attributes are described as follows: ● **PGDBNAME**: string type. This parameter specifies the database name. (This parameter does not need to be set in the URL. The system automatically parses the URL to obtain its value.) ● **PGHOST**: string type. This parameter specifies the host IP address. For details, see the example below. ● **PGPORT**: integer type. This parameter specifies the host port number. For details, see the example below. ● **user**: string type. This parameter specifies the database user who creates the connection. ● **password**: string type. This parameter specifies the password of the database user. ● **enable_ce**: string type. If **enable_ce** is set to **1**, JDBC supports encrypted equality query. ● **refreshClientEncryption**: string type. If **refreshClientEncryption** is set to **1** (default value), the encrypted database supports cache update on the client. ● **loggerLevel**: string type. The following log levels are supported: **OFF**, **DEBUG**, and **TRACE**. The value **OFF** indicates that the log function is disabled. **DEBUG** and **TRACE** logs record information of different levels. ● **loggerFile**: string type. This parameter specifies the name of a log file. You can specify a directory for storing logs. If no directory is specified, logs are stored in the directory where the client program is running. This parameter has been discarded and does not take effect. To use this parameter, you can configure it in the **java.util.logging** attribute file or system attributes. ● **allowEncodingChanges**: Boolean type. If this parameter is set to **true**, the character set type can be changed. This parameter is used together with **characterEncoding=CHARSET** to set the character set. The two parameters are separated by ampersands (&). The value of **characterEncoding** can be **UTF8**, **GBK**, or **LATIN1**. ● **currentSchema**: string type. This parameter specifies the schema to be set in **search-path**. ● **hostRecheckSeconds**: integer type. After JDBC attempts to connect to a host, the host status is saved: connection success or connection failure. This status is trusted within the duration specified by **hostRecheckSeconds**. After the duration expires, the status becomes invalid. The default value is 10 seconds. ● **ssl**: Boolean type. This parameter specifies a connection in SSL mode. When **ssl** is set to **true**, the NonValidatingFactory channel and certificate mode are supported.

Parameter	Description
	1. For the NonValidatingFactory channel, configure the username and password and set **SSL** to **true**. 2. In certification mode, configure the client certificate, key, and root certificate, and set **SSL** to **true**. ● **sslmode**: string type. This parameter specifies the SSL authentication mode. The value can be **require**, **verify-ca**, or **verify-full**. – **require**: The system attempts to set up an SSL connection. If there is a CA file, the system performs verification as if the parameter was set to **verify-ca**. – **verify-ca**: The system attempts to set up an SSL connection and checks whether the server certificate is issued by a trusted CA. – **verify-full**: The system attempts to set up an SSL connection, checks whether the server certificate is issued by a trusted CA, and checks whether the host name of the server is the same as that in the certificate. ● **sslcert**: string type. This parameter specifies the complete path of the certificate file. The type of the client and server certificates is **End Entity**. ● **sslkey**: string type. This parameter specifies the complete path of the key file. You must run the following command to convert the client certificate to the DER format: `openssl pkcs8 -topk8 -outform DER -in client.key -out client.key.pk8 -nocrypt` ● **sslrootcert**: string type. This parameter specifies the name of the SSL root certificate. The root certificate type is CA. ● **sslpassword**: string type. This parameter is provided for ConsoleCallbackHandler. ● **sslpasswordcallback**: string type. This parameter specifies the class name of the SSL password provider. The default value is **org.postgresql.ssl.jdbc4.LibPQFactory.ConsoleCallbackHandler**. ● **sslfactory**: string type. This parameter specifies the class name used by SSLSocketFactory to establish an SSL connection. ● **sslfactoryarg**: string type. The value is an optional parameter of the constructor function of the **sslfactory** class and is not recommended. ● **sslhostnameverifier**: string type. This parameter specifies the class name of the host name verifier. The API must implement javax.net.ssl.HostnameVerifier. The default value is **org.postgresql.ssl.PGjdbcHostnameVerifier**. ● **loginTimeout**: integer type. This parameter specifies the waiting time for establishing the database connection, in seconds. When multiple IP addresses are configured in the URL, if the time for obtaining the connection exceeds the value of this parameter, the connection fails and the subsequent IP addresses are not tried.

Parameter	Description
	● **connectTimeout**: integer type. This parameter specifies the timeout duration for connecting to a server, in seconds. If the time taken to connect to a server exceeds the value specified, the connection is interrupted. If the value is **0**, the timeout mechanism is disabled. When multiple IP addresses are configured in the URL, this parameter indicates the timeout interval for connecting to a single IP address. ● **socketTimeout**: integer type. This parameter specifies the timeout duration for a socket read operation, in seconds. If the time taken to read data from a server exceeds the value specified, the connection is closed. If the value is **0**, the timeout mechanism is disabled. ● **cancelSignalTimeout**: integer type. Cancel messages may cause a block. This parameter controls **connectTimeout** and **socketTimeout** in a cancel message, in seconds. The default value is 10 seconds. ● **tcpKeepAlive**: Boolean type. This parameter is used to enable or disable TCP keepalive detection. The default value is **false**. ● **logUnclosedConnections**: Boolean type. The client may leak a connection object because it does not call the connection object's close() method. These objects will be collected as garbage and finalized using the finalize() method. If the caller ignores this operation, this method closes the connection. ● **assumeMinServerVersion**: string type. The client sends a request to set a floating point. This parameter specifies the version of the server to connect, for example, **assumeMinServerVersion=9.0**. This parameter can reduce the number of packets to send during connection setup. ● **ApplicationName**: string type. This parameter specifies the name of the JDBC driver that is being connected. You can query the **pg_stat_activity** table on the primary database node to view information about the client that is being connected. The JDBC driver name is displayed in the **application_name** column. The default value is **PostgreSQL JDBC Driver**. ● **connectionExtraInfo**: Boolean type. This parameter specifies whether the JDBC driver reports the driver deployment path and process owner to the database. The value can be **true** or **false**. The default value is **false**. If **connectionExtraInfo** is set to **true**, the JDBC driver reports the driver deployment path, process owner, and URL connection configuration information to the database and displays the information in the **connection_info** parameter. In this case, you can query the information from **PG_STAT_ACTIVITY**. ● **autosave**: string type. The value can be **always**, **never**, or **conservative**. The default value is **never**. This parameter specifies the action that the driver should perform upon a query failure. If **autosave** is set to **always**, the JDBC driver sets a savepoint before

Parameter	Description
	each query and rolls back to the savepoint if the query fails. If **autosave** is set to **never**, there is no savepoint. If **autosave** is set to **conservative**, a savepoint is set for each query. However, the system rolls back and retries only when there is an invalid statement. ● **protocolVersion**: integer type. This parameter specifies the connection protocol version. Only versions 1 and 3 are supported. Note: If this parameter is set to **1**, only the V1 server is connected. MD5 encryption is used when this parameter is set to **3**. You must need to set **password_encryption_type** to **1** to change the database encryption mode. After the database is restarted, create a user that uses MD5 encryption to encrypt passwords. You must also change the client connection mode to **md5** in the **pg_hba.conf** file. Log in as the new user (not recommended). **NOTE** The MD5 encryption algorithm has lower security and poses security risks. Therefore, you are advised to use a more secure encryption algorithm. ● **prepareThreshold**: integer type. This parameter specifies the time when the parse statement is sent. The default value is **5**. It takes a long time to parse an SQL statement for the first time, but a short time to parse SQL statements later because of cache. If a session runs an SQL statement multiple consecutive times and the number of execution times exceeds the value of **prepareThreshold**, JDBC does not send the parse command to the SQL statement. ● **preparedStatementCacheQueries**: integer type. This parameter specifies the number of queries cached in each connection. The default value is **256**. If more than 256 different queries are used in the prepareStatement() call, the least recently used query cache will be discarded. The value **0** indicates that the cache function is disabled. ● **preparedStatementCacheSizeMiB**: integer type. This parameter specifies the maximum cache size of each connection, in MB. The default value is **5**. If the size of the cached queries exceeds 5 MB, the least recently used query cache will be discarded. The value **0** indicates that the cache function is disabled. ● **databaseMetadataCacheFields**: integer type. The default value is **65536**. This parameter specifies the maximum cache size of each connection. The value **0** indicates that the cache function is disabled. ● **databaseMetadataCacheFieldsMiB**: integer type. The default value is **5**. This parameter specifies the maximum cache size of each connection, in MB. The value **0** indicates that the cache function is disabled. ● stringtype: string type. The value can be **false**, **unspecified**, or **varchar**. The default value is **varchar**. This parameter specifies the type of the **PreparedStatement** parameter used by the setString()

Parameter	Description
	method. If **stringtype** is set to **varchar**, these parameters are sent to the server as varchar parameters. If **stringtype** is set to **unspecified**, these parameters are sent to the server as an untyped value, and the server attempts to infer their appropriate type. ● **batchMode**: string type. This parameter specifies whether to connect the database in batch mode. The default value is **on**, indicating that the batch mode is enabled. ● **fetchsize**: integer type. This parameter specifies the default fetchsize for statements in the created connection. The default value is **0**, indicating that all results are obtained at a time. It is equivalent to defaultRowFetchSize. ● **reWriteBatchedInserts**: Boolean type. During batch import, set this parameter to **true** to combine *N* insertion statements into one: **insert into TABLE_NAME values(values1, ..., valuesN), ..., (values1, ..., valuesN)**. To use this parameter, set **batchMode** to **off**. ● **unknownLength**: integer type. The default value is **Integer.MAX_VALUE**. This parameter specifies the length of the unknown length type when the data of some postgresql types (such as TEXT) is returned by functions such as ResultSetMetaData.getColumnDisplaySize and ResultSetMetaData.getPrecision. ● **uppercaseAttributeName**: Boolean type. The default value is **false**, indicating that the function is disabled. The value **true** indicates that the function is enabled. If this parameter is enabled, the query result of the API for obtaining metadata is converted to uppercase letters. The application scenario is as follows: All metadata stored in the database is in lowercase, but the metadata in uppercase must be used as the input and output parameters. Involved APIs: **java.sql.DatabaseMetaData** and **java.sql.ResultSetMetaData** ● **defaultRowFetchSize**: integer type. This parameter specifies the number of rows read by fetch in ResultSet at a time. Limiting the number of rows read each time in a database access request can avoid unnecessary memory consumption, thereby avoiding out of memory exception. The default value is **0**, indicating that all rows are obtained at a time in ResultSet. There is no negative value. ● **binaryTransfer**: Boolean type. This parameter specifies whether data is sent and received in binary format. The default value is **false**. ● **binaryTransferEnable**: string type. This parameter specifies the type for which binary transmission is enabled. Every two types are separated by commas (,). You can select either the OID or name, for example, binaryTransferEnable=Integer4_ARRAY,Integer8_ARRAY. For example, if the OID name is **BLOB** and the OID number is **88**, you can configure the OID as follows:

Parameter	Description
	binaryTransferEnable=BLOB or **binaryTransferEnable=88** ● **binaryTransferDisEnable**: string type. This parameter specifies the type for which binary transmission is disabled. Every two types are separated by commas (,). You can select either the OID or name. The value of this parameter overwrites the value of **binaryTransferEnable**. ● **blobMode**: string type. This parameter sets the setBinaryStream method to assign values to different types of data. The value **on** indicates that values are assigned to blob data. The value **off** indicates that values are assigned to bytea data. The default value is **on**. ● **socketFactory**: string type. This parameter specifies the name of the class used to create a socket connection with the server. This class must implement the **javax.net.SocketFactory** API and define a constructor with no parameter or a single string parameter. ● **socketFactoryArg**: string type. The value is an optional parameter of the constructor function of the socketFactory class and is not recommended. ● **receiveBufferSize**: integer type. This parameter is used to set **SO_RCVBUF** on the connection stream. ● **sendBufferSize**: integer type. This parameter is used to set **SO_SNDBUF** on the connection stream. ● **preferQueryMode**: string type. The value can be **extended**, **extendedForPrepared**, **extendedCacheEverything**, or **simple**. This parameter specifies the query mode. In **simple** mode, the query is executed without parsing or binding. In **extended** mode, the query is executed and bound. The **extendedForPrepared** mode is used for prepared statement extension. In **extendedCacheEverything** mode, each statement is cached. ● **targetServerType**: string type. This parameter is used to identify the primary DN and standby DN by querying whether a DN allows the write operation in the URL connection string. The default value is **any**. The value can be **any**, **master**, **slave**, or **preferSlave**. – **master**: attempts to connect to a primary DN in the URL connection string. If the primary DN cannot be found, an exception is thrown. – **slave**: attempts to connect to a standby DN in the URL connection string. If the standby DN cannot be found, an exception is thrown. – **preferSlave**: attempts to connect to a standby DN (if available) in the URL connection string. Otherwise, it connects to the primary DN. – **any**: attempts to connect to any DN in the URL connection string. ● **priorityServers**: integer type. This value is used to specify the first *n* nodes configured in the URL as the primary database instance to

Parameter	Description
	be connected preferentially. The default value is **null**. The value is a number greater than 0 and less than the number of DNs configured in the URL. It is used in streaming DR scenarios. For example, jdbc:postgresql:// host1:port1,host2:port2,host3:port3,host4:port4,/database? priorityServers=2. That is, **host1** and **host2** are primary database instance nodes, and **host3** and **host4** are DR database instance nodes. ● **forceTargetServerSlave**: Boolean type. This parameter specifies whether to enable the function of forcibly connecting to the standby node and forbid the existing connections to be used on the standby node that is promoted to primary during the primary/ standby switchover of the database instance. The default value is **false**, indicating that the function of forcibly connecting to the standby node is disabled. **true**: The function of forcibly connecting to the standby node is enabled. ● **iamUser**: string type. The fully-encrypted database encrypts data on the client. The current feature is a lab feature. Contact Huawei technical support before using it. During encryption, you can access the Key Management Service (KMS) provided by Huawei Cloud to obtain keys. When accessing the KMS, you need to provide the IAM identity authentication information and KMS project information. **iamUser** and **iamPassword** are used to set identity authentication information. **kmsDomain**, **kmsProjectId**, and **kmsProjectName** are used to set KMS project information. To obtain the preceding five parameters, log in to the Huawei Cloud official website and choose **Console** > **My Credential**. ● **iamPassword**: string type, used to set the password of the IAM user. ● **kmsDoamin**: string type, used to set the Huawei Cloud account to which the KMS belongs. ● **kmsProjectName**: string type, used to set the deployment zone of a KMS project. KMS projects deployed in different zones are isolated from each other. ● **kmsProjectId**: string type, used to set the ID of a KMS project. ● **traceInterfaceClass**: String type. The default value is **null**, which is used to obtain the implementation class of traceId. The value is the fully qualified name of the implementation class of the **org.postgresql.log.Tracer** API that implements the method for obtaining traceId. ● **use_boolean**: Boolean type. This parameter is used to set the OID type bound to the setBoolean method in extended mode. The default value is **false**, indicating that the int2 type is bound. The value **true** indicates that the Boolean type is bound. ● **allowReadOnly**: Boolean type. This parameter is used to determine whether the read-only mode is allowed. The default

Parameter	Description
	value is **true**, indicating that the read-only mode is allowed. If this parameter is set to **false**, the read-only mode is disabled. ● **TLSCiphersSupperted**: String type. This parameter is used to set the supported TLS encryption suite. The default value is **TLS_DHE_RSA_WITH_AES_128_GCM_SHA256,TLS_DHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384**. ● **stripTrailingZeros**: Boolean type. The default value is **false**. If this parameter is set to **true**, trailing 0s of the numeric type are removed. This attribute is valid only for ResultSet.getObject(int columnIndex). ● **enableTimeZone**: Boolean type. The default value is **true**. This parameter specifies whether to enable the time zone setting on the server. The value **true** indicates that the JVM time zone is obtained to specify the database time zone. The value **false** indicates that the database time zone is used. ● **loadBalanceHosts**: Boolean type. In the default mode (disabled), multiple hosts specified in the URL are connected in default sequence. If load balancing is enabled, the shuffle algorithm is used to randomly select a host from the candidate hosts to establish a connection. In a centralized environment, ensure that no write operation is in services when using this parameter. ● **socketTimeoutInConnecting**: Integer type. The default value is **5s**. It specifies the timeout value for a socket read operation during connection establishment. If the time taken to read data from a server exceeds the value specified during connection establishment, the connection is closed. If the value is **0**, the timeout mechanism is disabled.
user	Database user.
password	Password of the database user.

NOTE

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

Examples

```
// The following code encapsulates database connection operations into an API. The database can then be connected using an authorized username and a password.
public static Connection getConnect(String username, String passwd)
   {
        // Driver class.
        String driver = "org.postgresql.Driver";
        // Database connection descriptor.
        String sourceURL = "jdbc:postgresql://10.10.0.13:8000/postgres";
        Connection conn = null;

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

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

        return conn;
    }
// The following code uses the Properties object as a parameter to establish a connection.
public static Connection getConnectUseProp(String username, String passwd)
    {
        // Driver class.
        String driver = "org.postgresql.Driver";
        // Database connection descriptor.
        String sourceURL = "jdbc:postgresql://10.10.0.13:8000/postgres?";
        Connection conn = null;
        Properties info = new Properties();

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

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

        return conn;
    }
```

5.3.5 Connecting to the Database (Using SSL)

When establishing connections to the GaussDB server using JDBC, you can enable SSL connections to encrypt client and server communications for security of sensitive data transmission on the Internet. This section describes how applications establish an SSL connection to GaussDB using JDBC. To start the SSL mode, you must have the server certificate, client certificate, and private key files. For details about how to obtain these files, see related documents and commands of OpenSSL.

Configuring the Client

To configure the client, perform the following steps:

Upload the certificate files **client.key.pk8**, **client.crt**, and **cacert.pem** to the client.

Examples

Note: Select either example 1 or example 2.

```
public class SSL{
    public static void main(String[] args) {
        Properties urlProps = new Properties();
        String urls = "jdbc:postgresql://10.29.37.136:8000/postgres";

        /**
         * ================== Example 1: Use the NonValidatingFactory channel.
         */
        urlProps.setProperty("sslfactory","org.postgresql.ssl.NonValidatingFactory");
        urlProps.setProperty("user", "world");
        urlProps.setProperty("password", "test@123");
        urlProps.setProperty("ssl", "true");
        /**
         * ==================  Examples 2: Use a certificate.
         */
        urlProps.setProperty("sslcert", "client.crt");
        urlProps.setProperty("sslkey", "client.key.pk8");
        urlProps.setProperty("sslrootcert", "cacert.pem");
        urlProps.setProperty("user", "world");
        urlProps.setProperty("ssl", "true");
        /* sslmode can be set to require, verify-ca, or verify-full. Select one from the following three 
examples.*/
       /* ================== Example 2.1: Set sslmode to require to use the certificate for authentication. 
*/
        urlProps.setProperty("sslmode", "require");
       /* ================== Example 2.2: Set sslmode to verify-ca to use the certificate for 
authentication. */
        urlProps.setProperty("sslmode", "verify-ca");
       /* ================== Example 2.3: Set sslmode to verify-full to use the certificate (in the Linux 
OS) for authentication. */
        urls = "jdbc:postgresql://world:8000/postgres";
        urlProps.setProperty("sslmode", "verify-full");
        try {
            Class.forName("org.postgresql.Driver").newInstance();
        } catch (Exception e) {
            e.printStackTrace();
        }
        try {
            Connection conn;
            conn = DriverManager.getConnection(urls,urlProps);
            conn.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
/**
 * Note: Convert the client key to the DER format.
 * openssl pkcs8 -topk8 -outform DER -in client.key -out client.key.pk8 -nocrypt
 * openssl pkcs8 -topk8 -inform PEM -in client.key -outform DER -out client.key.der -v1 PBE-MD5-DES
 * openssl pkcs8 -topk8 -inform PEM -in client.key -outform DER -out client.key.der -v1 PBE-SHA1-3DES
 * The preceding algorithms are not recommended due to their low security.
 * If the customer needs to use a higher-level private key encryption algorithm, the following private key
encryption algorithms can be used after the BouncyCastle or a third-party private key is used to decrypt the
password package:
 * openssl pkcs8 -in client.key -topk8  -outform DER -out client.key.der -v2 AES128
 * openssl pkcs8 -in client.key -topk8  -outform DER -out client.key.der -v2 aes-256-cbc -iter 1000000
 * openssl pkcs8 -in client.key -topk8 -out client.key.der  -outform Der -v2 aes-256-cbc -v2prf
hmacWithSHA512
 * Enable BouncyCastle: Introduce the bcpkix-jdk15on.jar package for projects that use JDBC. The
recommended version is 1.65 or later.
 */
```

5.3.6 Connecting to a Database (Using UDS)

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

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

Example:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
import java.util.Properties;

public class Test {
    public static void main(String[] args) {
        String driver = "org.postgresql.Driver";
        Connection conn;
        try {
            Class.forName(driver).newInstance();
            Properties properties = new Properties();
            properties.setProperty("user", "username");
            properties.setProperty("password", "password");
            conn = DriverManager.getConnection("jdbc:postgresql://localhost:8000/postgres?
socketFactory=org.newsclub" +
                            ".net.unix" +
                            ".AFUNIXSocketFactory$FactoryArg&socketFactoryArg=/data/tmp/.s.PGSQL.8000",
                    properties);
            System.out.println("Connection Successful!");
            Statement statement = conn.createStatement();
            statement.executeQuery("select 1");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

NOTICE

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

5.3.7 Running SQL Statements

Running a Common SQL Statement

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

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

```
Connection conn = DriverManager.getConnection("url","user","password");
Statement stmt = conn.createStatement();
```

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

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

NOTE

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

Step 3 Close the statement object.

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

----End

Running a Prepared SQL Statement

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

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

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

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

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

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

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

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

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

----End

Calling a Stored Procedure

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

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

```
Connection myConn = DriverManager.getConnection("url","user","password");
CallableStatement cstmt = myConn.prepareCall("{? = CALL TESTPROC(?,?,?)}");
```

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

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

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

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

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

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

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

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

Example:

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

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

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

NOTE

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

NOTICE

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

----End

Calling a Stored Procedure When Overloading Is Enabled in Oracle Compatibility Mode

After the **behavior_compat_options='proc_outparam_override'** parameter is enabled, perform the following steps to call the stored procedure:

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

```
Connection conn = DriverManager.getConnection("url","user","password");
CallableStatement cs = conn.prepareCall("{ CALL TEST_PROC(?,?,?) }");
```

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

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

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

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

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

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

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

```
PGobject result = (PGobject)cs.getObject(2);  // Obtain the out parameter.
result.getValue(); // Obtain the string value of the composite type.
result.getArrayValue(); // Obtain the array values of the composite type and sort the values according to
the sequence of columns of the composite type.
result.getStruct(); // Obtain the subtype names of the composite type and sort them according to the
creation sequence.
```

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

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

NOTE

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

----End

Example:

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

Batch Processing

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

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

```
Connection conn = DriverManager.getConnection("url","user","password");
PreparedStatement pstmt = conn.prepareStatement("INSERT INTO customer_t1 VALUES (?)");
```

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

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

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

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

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

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

NOTE

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

----End

5.3.8 Processing Data in a Result Set

Setting a Result Set Type

Different types of result sets apply to different application scenarios. Applications select proper types of result sets based on requirements. Before running an SQL statement, you must create a statement object. Some methods of creating statement objects can set the type of a result set. **Table 5-5** lists result set parameters. The related Connection methods are as follows:

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

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

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

Table 5-5 Result set types

Parameter	Description
resultSetType	Type of a result set. There are three types of result sets: ● **ResultSet.TYPE_FORWARD_ONLY**: The ResultSet object can only be navigated forward. It is the default value. ● **ResultSet.TYPE_SCROLL_SENSITIVE**: You can view the modified result by scrolling to the modified row. ● **ResultSet.TYPE_SCROLL_INSENSITIVE**: The ResultSet object is insensitive to changes in the underlying data source. **NOTE** After a result set has obtained data from the database, the result set is insensitive to data changes made by other transactions, even if the result set type is **ResultSet.TYPE_SCROLL_SENSITIVE**. To obtain up-to-date data of the record pointed by the cursor from the database, call the refreshRow() method in a ResultSet object.

Parameter	Description
resultSetConcurren-cy	Concurrency type of a result set. There are two types of concurrency. ● **ResultSet.CONCUR_READ_ONLY**: Data in a result set cannot be updated except that an updated statement has been created in the result set data. ● **ResultSet.CONCUR_UPDATEABLE**: changeable result set. The concurrency type for a result set object can be updated if the result set is scrollable.

Positioning a Cursor in a Result Set

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

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

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

Obtaining the Cursor Position from a Result Set

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

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

Method	Description
isFirst()	Checks whether the cursor is in the first row.
isLast()	Checks whether the cursor is in the last row.
isBeforeFirst()	Checks whether the cursor is before the first row.
isAfterLast()	Checks whether the cursor is after the last row.
getRow()	Gets the current row number of the cursor.

Obtaining Data from a Result Set

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

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

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

Method	Description
Date getDate(String columnLabel)	Retrieves the value of the column designated by a column name in the current row as a date.

5.3.9 Closing a Connection

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

Call the close method to close the connection, for example, **Connection conn = DriverManager.getConnection("url","user","password"); conn.close();**

5.3.10 Log Management

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

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

SLF4J and JdkLogger are mainstream frameworks for Java application log management in the industry. For details about how to use these frameworks, see the official documents (SLF4J: **http://www.slf4j.org/manual.html**; JdkLogger: **https://docs.oracle.com/javase/8/docs/technotes/guides/logging/overview.html**).

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

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

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

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

⚠ CAUTION

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

Example:

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

    String sourceURL = "jdbc:postgresql://10.10.0.13:8000/postgres?logger=Slf4JLogger";
    Connection conn = null;

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

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

```
log4j.logger.org.postgresql=ALL, log_gsjdbc

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

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

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

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

The default Java logging framework stores its configurations in a file named **logging.properties**. Java installs the global configuration file in the folder in the

Java installation directory. The **logging.properties** file can also be created and stored with a single project.

Configuration example of **logging.properties**:

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

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

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

The following is a code example:

```
System.setProperty("java.util.logging.FileHandler.pattern","jdbc.log");
FileHandler fileHandler = new FileHandler(System.getProperty("java.util.logging.FileHandler.pattern"));
Formatter formatter = new SimpleFormatter();
fileHandler.setFormatter(formatter);
Logger logger = Logger.getLogger("org.postgresql");
logger.addHandler(fileHandler);
logger.setLevel(Level.ALL);
logger.setUseParentHandlers(false);
```

Link Trace Function

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

URL example:

```
String URL = "jdbc:postgresql://127.0.0.1:8000/postgres?
traceInterfaceClass=xxx.xxx.xxx.OpenGaussTraceImpl";
```

The **org.postgresql.log.Tracer** API class is defined as follows:

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

The following is an example of the **org.postgresql.log.Tracer** API implementation class:

```
import org.postgresql.log.Tracer;

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

    private final String TRACE_ID_KEY = "traceId";

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

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

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

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

```
import java.util.HashMap;

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

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

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

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

The following is an example of using traceId:

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

NOTE

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

5.3.11 Examples: Common Operations

Example 1:

This example illustrates how to develop applications based on the JDBC API provided by GaussDB. Before executing the code in this example, load the driver first. For details about how to obtain and load the driver, see **JDBC Package, Driver Class, and Environment Class**.

```
//DBtest.java
// This example illustrates the main processes of JDBC-based development, covering database connection
creation, table creation, and data insertion.

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.CallableStatement;
import java.sql.Types;

public class DBTest {

  // Create a database connection.
  public static Connection GetConnection(String username, String passwd) {
    String driver = "org.postgresql.Driver";
    String sourceURL = "jdbc:postgresql://localhost:8000/postgres";
    Connection conn = null;
    try {
      // Load the database driver.
      Class.forName(driver).newInstance();
    } catch (Exception e) {
      e.printStackTrace();
      return null;
    }

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

    return conn;
  };

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

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

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

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

    try {
      // Generate a prepared statement.
      pst = conn.prepareStatement("INSERT INTO customer_t1 VALUES (?,?)");
      for (int i = 0; i < 3; i++) {
        // Add parameters.
        pst.setInt(1, i);
        pst.setString(2, "data " + i);
        pst.addBatch();
      }
      // Perform batch processing.
      pst.executeBatch();
      pst.close();
    } catch (SQLException e) {
      if (pst != null) {
        try {
          pst.close();
        } catch (SQLException e1) {
        e1.printStackTrace();
        }
      }
      e.printStackTrace();
    }
  }

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


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


  /**
   * Main process. Call static methods one by one.
   * @param args
  */
  public static void main(String[] args) {
  // Create a database connection.
    Connection conn = GetConnection("tester", "Password1234");

    // Create a table.
    CreateTable(conn);

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

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

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

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

  }

}
```

Example 2: High Client Memory Usage

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

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

```
// Disable automatic commit.
conn.setAutoCommit(false);
Statement st = conn.createStatement();

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

// Disable the server cursor.
```

```
st.setFetchSize(0);
rs = st.executeQuery("SELECT * FROM mytable");
while (rs.next())
{
    System.out.print("many rows were returned.");
}
conn.commit();
rs.close();

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

Run the following command to enable automatic commit:

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

Example 3: Example of Common Data Types

```
//Example of the bit type. Note that the value range of the bit type is [0,1].
Statement st = conn.createStatement();
String sqlstr = "create or replace function fun_1()\n" +
        "returns bit AS $$\n" +
        "select col_bit from t_bit limit 1;\n" +
        "$$\n" +
        "LANGUAGE SQL;";
st.execute(sqlstr);
CallableStatement c = conn.prepareCall("{ ? = call fun_1() }");
//Register the output type, which is a bit string.
c.registerOutParameter(1, Types.BIT);
c.execute();
//Use the Boolean type to obtain the result.
System.out.println(c.getBoolean(1));

// Example of using the money type
// Example of using a column of the money type in the table structure.
st.execute("create table t_money(col1 money)");
PreparedStatement pstm = conn.prepareStatement("insert into t_money values(?)");
// Use PGobject to assign a value. The value range is [-92233720368547758.08,92233720368547758.07].
PGobject minMoney = new PGobject();
minMoney.setType("money");
minMoney.setValue("-92233720368547758.08");
pstm.setObject(1, minMoney);
pstm.execute();
// Use PGMoney to assign a value. The value range is [-9999999.99,9999999.99].
pstm.setObject(1,new PGmoney(9999999.99));
pstm.execute();

// Example of using the function whose return value is money.
st.execute("create or replace function func_money() " +
                "return money " +
                "as declare " +
                "var1 money; " +
                "begin " +
                " select col1 into var1 from t_money limit 1; " +
                " return var1; " +
                "end;");
CallableStatement cs = conn.prepareCall("{? = call func_money()}");
cs.registerOutParameter(1,Types.DOUBLE);
cs.execute();
cs.getObject(1);
```

Example 4: Obtaining the Driver Version

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

5.3.12 Example: Retrying SQL Queries for Applications

If the primary database node is faulty and cannot be restored within 10s, GaussDB automatically promotes the standby database node to primary to ensure that the database runs properly. Jobs running during the failover will fail and those started after the failover will not be affected. To prevent upper-layer services from being affected by the failover, refer to the following example to construct an SQL retry mechanism at the service layer. Before executing the code in this example, load the driver first. For details about how to obtain and load the driver, see **JDBC Package, Driver Class, and Environment Class**.

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

class ExitHandler extends Thread {
    private Statement cancel_stmt = null;

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

public class SQLRetry {
   // Create a database connection.
   public static Connection GetConnection(String username, String passwd) {
     String driver = "org.postgresql.Driver";
     String sourceURL = "jdbc:postgresql://10.131.72.136:8000/postgres";
     Connection conn = null;
     try {
       // Load the database driver.
       Class.forName(driver).newInstance();
     } catch (Exception e) {
       e.printStackTrace();
       return null;
     }

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

     return conn;
}

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

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

      // Run a common SQL statement.
      int rc2 = stmt
         .executeUpdate("DROP TABLE if exists jdbc_test1;");

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

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

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

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

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

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

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

      pstmt.close();
```

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

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

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

  }

  /**
   *Main process. Call static methods one by one.
   * @param args
   * @throws InterruptedException
   */
  public static void main(String[] args) throws InterruptedException {
    // Create a database connection.
    Connection conn = GetConnection("testuser", "test@123");

    // Create a table.
    CreateTable(conn);

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

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

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

  }

}
```

5.3.13 Example: Importing and Exporting Data Through Local Files

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

The sample program is as follows. Load the driver before executing the sample code. For details about how to obtain and load the driver, see **JDBC Package, Driver Class, and Environment Class**.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.io.IOException;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.sql.SQLException;
import org.postgresql.copy.CopyManager;
import org.postgresql.core.BaseConnection;

public class Copy{

    public static void main(String[] args)
    {
     String urls = new String("jdbc:postgresql://10.180.155.74:8000/postgres"); // Database URL
     String username = new String("jack");            // Username
     String password = new String("xxxxxxxxx");             // Password
     String tablename = new String("migration_table"); // Table information
     String tablename1 = new String("migration_table_1"); // Table information
     String driver = "org.postgresql.Driver";
     Connection conn = null;

     try {
           Class.forName(driver);
           conn = DriverManager.getConnection(urls, username, password);
         } catch (ClassNotFoundException e) {
              e.printStackTrace(System.out);
         } catch (SQLException e) {
              e.printStackTrace(System.out);
         }

     // Export the query result of SELECT * FROM migration_table to the local file d:/data.txt.
     try {
    copyToFile(conn, "d:/data.txt", "(SELECT * FROM migration_table)");
  } catch (SQLException e) {

 e.printStackTrace();
  } catch (IOException e) {

 e.printStackTrace();
  }
     // Import data from the d:/data.txt file to the migration_table_1 table.
     try {
     copyFromFile(conn, "d:/data.txt", tablename1);
  } catch (SQLException e) {
        e.printStackTrace();
} catch (IOException e) {

 e.printStackTrace();
}

     // Export the data from the migration_table_1 table to the d:/data1.txt file.
     try {
     copyToFile(conn, "d:/data1.txt", tablename1);
  } catch (SQLException e) {

 e.printStackTrace();
```

```
  } catch (IOException e) {


 e.printStackTrace();
}
    }
 // Use copyIn to import data from a file to the database.
 public static void copyFromFile(Connection connection, String filePath, String tableName)
        throws SQLException, IOException {

    FileInputStream fileInputStream = null;

    try {
        CopyManager copyManager = new CopyManager((BaseConnection)connection);
        fileInputStream = new FileInputStream(filePath);
        copyManager.copyIn("COPY " + tableName + " FROM STDIN", fileInputStream);
    } finally {
        if (fileInputStream != null) {
            try {
                fileInputStream.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}

 // Use copyOut to export data from the database to a file.
 public static void copyToFile(Connection connection, String filePath, String tableOrQuery)
         throws SQLException, IOException {

     FileOutputStream fileOutputStream = null;

     try {
         CopyManager copyManager = new CopyManager((BaseConnection)connection);
         fileOutputStream = new FileOutputStream(filePath);
         copyManager.copyOut("COPY " + tableOrQuery + " TO STDOUT", fileOutputStream);
     } finally {
         if (fileOutputStream != null) {
             try {
                 fileOutputStream.close();
             } catch (IOException e) {
                 e.printStackTrace();
             }
         }
     }
 }
}
```

5.3.14 Example: Migrating Data from MY

The following example shows how to use CopyManager to migrate data from MY to GaussDB. Before executing the code in this example, load the driver first. For details about how to obtain and load the driver, see **JDBC Package, Driver Class, and Environment Class**.

```
import java.io.StringReader;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import org.postgresql.copy.CopyManager;
import org.postgresql.core.BaseConnection;

public class Migration{
    public static void main(String[] args) {
        String url = new String("jdbc:postgresql://10.180.155.74:8000/postgres"); // Database URL
        String user = new String("jack");            //GaussDB username
        String pass = new String("xxxxxxxxx");             //GaussDB password
      String tablename = new String("migration_table"); // Table information
        String delimiter = new String("|");              // Delimiter
        String encoding = new String("UTF8");            // Character set
        String driver = "org.postgresql.Driver";
        StringBuffer buffer = new StringBuffer();       // Buffer to store formatted data

        try {
            // Obtain the query result set of the source database.
            ResultSet rs = getDataSet();

            // Traverse the result set and obtain records row by row.
            // The values of columns in each record are separated by the specified delimiter and end with a
linefeed, forming strings.
            // Add the strings to the buffer.
            while (rs.next()) {
                buffer.append(rs.getString(1) + delimiter
                        + rs.getString(2) + delimiter
                        + rs.getString(3) + delimiter
                        + rs.getString(4)
                        + "\n");
            }
            rs.close();

            try {
                // Connect to the target database.
                Class.forName(driver);
                Connection conn = DriverManager.getConnection(url, user, pass);
                BaseConnection baseConn = (BaseConnection) conn;
                baseConn.setAutoCommit(false);

                // Initialize the table.
                String sql = "Copy " + tablename + " from STDIN DELIMITER " + "'" + delimiter + "'" + "
ENCODING " + "'" + encoding + "'";

                // Commit data in the buffer.
                CopyManager cp = new CopyManager(baseConn);
                StringReader reader = new StringReader(buffer.toString());
                cp.copyIn(sql, reader);
                baseConn.commit();
                reader.close();
                baseConn.close();
            } catch (ClassNotFoundException e) {
                e.printStackTrace(System.out);
            } catch (SQLException e) {
                e.printStackTrace(System.out);
            }

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    //********************************
    // Return the query result set from the source database.
    //*********************************
    private static ResultSet getDataSet() {
        ResultSet rs = null;
        try {
            Class.forName("com.MY.jdbc.Driver").newInstance();
            Connection conn = DriverManager.getConnection("jdbc:MY://10.119.179.227:3306/jack?
useSSL=false&allowPublicKeyRetrieval=true", "jack", "xxxxxxxxx");
            Statement stmt = conn.createStatement();
            rs = stmt.executeQuery("select * from migration_table");
        } catch (SQLException e) {
            e.printStackTrace();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return rs;
    }
}
```

5.3.15 Example: Logic Replication Code

The following example demonstrates how to use the logical replication function through the JDBC API.

For logical replication, in addition to the configuration items described in **Logical Decoding**, the following configuration items are added for JDBC stream decoding:

1. Decoding thread concurrency

   Set **parallel-decode-num** to specify the number of decoder threads for parallel decoding. The value is an integer ranging from 1 to 20. The value **1** indicates that decoding is performed based on the original serial logic. Other values indicate that parallel decoding is enabled. The default value is **1**. When this parameter is set to **1**, do not configure the following options: **decode-style**, **sending-batch**, and **parallel-queue-size**.

2. Decoding format

   Configure **decode-style** to specify the decoding format. The value can be **'j'**, **'t'** or **'b'** of the char type, indicating the JSON, text, or binary format, respectively. The default value is **'b'**, indicating binary decoding. This option is set only when parallel decoding is allowed and binary decoding is supported only in the parallel decoding scenario. For the JSON and text formats corresponding to the binary format, in the decoding result sent in batches, the uint32 consisting of the first four bytes of each decoding statement indicates the total number of bytes of the statement (the four bytes occupied by the uint32 are excluded, and **0** indicates that the decoding of this batch ends). The 8-byte uint64 indicates the corresponding LSN (**begin** corresponds to **first_lsn**, **commit** corresponds to **end_lsn**, and other values correspond to the LSN of the statement).

NOTE

The binary encoding rules are as follows:

1. The first four bytes represent the total number of bytes of the decoding result of statements following the statement-level delimiter letter P (excluded) or the batch end character F (excluded). If the value is **0**, the decoding of this batch ends.
2. The next eight bytes (uint64) indicate the corresponding LSN (**begin** corresponds to **first_lsn**, **commit** corresponds to **end_lsn**, and other values correspond to the LSN of the statement).
3. The following 1-byte letter can be **B**, **C**, **I**, **U**, or **D**, representing BEGIN, COMMIT, INSERT, UPDATE, or DELETE.
4. If the letter described in 3 is **B**:
   1. The following eight bytes (uint64) indicate the CSN.
   2. The next eight bytes (uint64) indicate **first_lsn**.
   3. (Optional) If the next 1-byte letter is **T**, the following four bytes (uint32) indicate the timestamp length for committing the transaction. The following characters with the same length are the timestamp character string.
   4. (Optional) If the next one-byte letter is **N**, the following four bytes (uint32) indicate the length of the transaction user name. The following characters with the same length are the transaction user name.
   5. Because there may still be a decoding statement subsequently, a 1-byte letter **P** or **F** is used as a separator between statements. **P** indicates that there are still decoded statements in this batch, and **F** indicates that this batch is completed.
5. If the letter described in 3 is **C**:
   1. (Optional) If the next 1-byte letter is **X**, the following eight bytes (uint64) indicate XID.
   2. (Optional) If the next 1-byte letter is **T**, the following four bytes (uint32) indicate the timestamp length. The following characters with the same length are the timestamp character string.
   3. When logs are sent in batches, decoding results of other transactions may still exist after a COMMIT log is decoded. If the next 1-byte letter is **P**, the batch still needs to be decoded. If the letter is **F**, the batch decoding ends.
6. If the letter described in 3 is **I**, **U**, or **D**:
   1. The following two bytes (uint16) indicate the length of the schema name.
   2. The schema name is read based on the preceding length.
   3. The following two bytes (uint16) indicate the length of the table name.
   4. The table name is read based on the preceding length.
   5. (Optional) If the next 1-byte letter is **N**, it indicates a new tuple. If the letter is **O**, it indicates an old tuple. In this case, the new tuple is sent first.
      1. The following two bytes (uint16) indicate the number of columns to be decoded for the tuple, which is recorded as **attrnum**.
      2. The following procedure is repeated for *attrnum* times.
         1. The next two bytes (uint16) indicate the length of the column name.
         2. The column name is read based on the preceding length.
         3. The following four bytes (uint32) indicate the OID of the current column type.
         4. The next four bytes (uint32) indicate the length of the value (stored in the character string format) in the current column. If the value is **0xFFFFFFFF**, it indicates null. If the value is **0**, it indicates a character string whose length is 0.
         5. The column value is read based on the preceding length.

6. Because there may still be a decoding statement after, if the next one-byte letter is **P**, it indicates that the batch still needs to be decoded, and if the next one-byte letter is **F**, it indicates that decoding of the batch ends.

3. Decoding only on the standby node

   Configure the **standby-connection** option to specify whether to perform decoding only on the standby node. The value is of the Boolean type (**0** or **1**). The value **true** (or **1**) indicates that only the standby node can be connected for decoding. When the primary node is connected for decoding, an error is reported and the system exits. The value **false** (or **0**) indicates that there is no restriction. The default value is **false** (**0**).

4. Batch sending

   Configure **sending-batch** to specify whether to send results in batches. The value is **0** or **1**. The value **0** indicates that decoding results are sent one by one. The value **1** indicates that decoding results are sent in batches when the accumulated size of decoding results reaches 1 MB. The default value is **0**. This parameter can be set only during parallel decoding. In the scenario where batch sending is enabled, if the decoding format is 'j' or 't', before each original decoding statement, a uint32 type is added indicating the length of the decoding result (excluding the current uint32 type), and a uint64 type is added, indicating the LSN corresponding to the current decoding result.

5. Length of the parallel decoding queue

   Configure **parallel-queue-size** to specify the length of the queue for interaction among parallel logical decoding threads. The value ranges from 2 to 1024 and must be a power of 2. The default value is **128**. The queue length is positively correlated with the memory usage during decoding.

6. Memory threshold for logical decoding

   The **max-txn-in-memory** configuration item specifies the memory threshold for caching the intermediate decoding result of a single transaction, in MB. The value ranges from 0 to 100. The default value is **0**, indicating that the memory usage is not controlled. The **max-reorderbuffer-in-memory** configuration item specifies the memory threshold for caching intermediate decoding results of all transactions, in GB. The value ranges from 0 to 100. The default value is **0**, indicating that the memory usage is not controlled. When the memory usage exceeds the threshold, intermediate decoding results are written into a temporary file during decoding, affecting the logic decoding performance.

7. Logical decoding sending timeout threshold

   The **sender-timeout** configuration item specifies the heartbeat timeout threshold between the database and client. If no message is received from the client within the period, the logic decoding stops and disconnects from the client. The unit is ms, and the value range is [0, 2147483647]. The default value depends on the value of **logical_sender_timeout**.

8. User blacklist options for logical decoding

   Use the user blacklist for logical decoding. The transaction operations of blacklisted users are filtered from the logic decoding result. The options are as follows:

   a. **exclude-userids**: specifies the OIDs of blacklisted users. Multiple OIDs are separated by commas (,). The system does not check whether the user OIDs exist.

b. **exclude-users**: specifies blacklisted user names. Multiple user names are separated by commas (,). **dynamic-resolution** specifies whether to dynamically parse and identify user names. If the decoding is interrupted because the user does not exist and the corresponding blocklisted user does not exist at the time when logs are generated, you can set **dynamic-resolution** to **true** or delete the user name from the blocklist to start decoding and continue to obtain logical logs.
c. **dynamic-resolution**: indicates whether to dynamically parse blacklisted user names. The default value is **true**. If the parameter is set to **false**, an error is reported and the logic decoding exits when the decoding detects that the user does not exist in blacklist **exclude-users**. If the parameter is set to **true**, decoding continues when it detects that the user does not exist in blacklist **exclude-users**.

9. Output options for transaction logic logs
   a. **include-xids**: indicates whether the BEGIN logical log of a transaction outputs the transaction ID. The default value is **true**.
   b. **include-timestamp**: indicates whether the BEGIN logical log of a transaction outputs the time when the transaction is committed. The default value is **false**.
   c. **include-user**: indicates whether the BEGIN logical log of a transaction outputs the user name of the transaction. The default value is **false**. The user name of a transaction refers to the authorized user, that is, the login user who executes the session corresponding to the transaction. The user name does not change during the execution of the transaction.
10. By default, **socketTimeout** of the logical decoding connection is set to **10s**. When the primary node is overloaded during decoding on the standby node, the connection may be closed due to timeout. You can set **withStatusInterval(10000,TimeUnit.MILLISECONDS)** to adjust the timeout interval.

The decoding performance (Xlog consumption) is greater than or equal to 100 MBps in the following standard parallel decoding scenario: 16-core CPU, 128 GB memory, network bandwidth > 200 MBps, 10 to 100 columns in a table, 0.1 KB to 1 KB data volume in a single row, DML operations are mainly INSERT operations, the number of statements in a single transaction is less than 4096, **parallel-decode-num** is set to **8**, the decoding format is **'b'**, and the batch sending function is enabled. To ensure that the decoding performance meets the requirements and minimize the impact on services, you are advised to set up only one parallel decoding connection on a standby node to ensure that the CPU, memory, and bandwidth resources are sufficient.

Note: The logical replication class PGReplicationStream is a non-thread-safe class. Concurrent calls may cause data exceptions. Before executing the code in this example, load the driver first. For details about how to obtain and load the driver, see **JDBC Package, Driver Class, and Environment Class**.

```
// Logical replication function example: file name, LogicalReplicationDemo.java
// Prerequisite: Add the IP address of the JDBC user machine to the database whitelist. Add the following
content to pg_hba.conf:
// Assume that the IP address of the JDBC user machine is 10.10.10.10.
//host    all            all            10.10.10.10/32        sha256
//host    replication     all            10.10.10.10/32        sha256

import org.postgresql.PGProperty;
import org.postgresql.jdbc.PgConnection;
import org.postgresql.replication.LogSequenceNumber;
import org.postgresql.replication.PGReplicationStream;

import java.nio.ByteBuffer;
import java.sql.DriverManager;
import java.util.Properties;
import java.util.concurrent.TimeUnit;

public class LogicalReplicationDemo {
    private static PgConnection conn = null;
    public static void main(String[] args) {
        String driver = "org.postgresql.Driver";
 // Configure the IP address and haPort number of the database. By default, the port number is the port
number of the connected DN plus 1.
        String sourceURL = "jdbc:postgresql://$ip:$port/postgres";
 // The default name of the logical replication slot is replication_slot.
 // Test mode: Create a logical replication slot.
        int TEST_MODE_CREATE_SLOT = 1;
 // Test mode: Enable logical replication (the prerequisite is that the logical replication slot already exists).
        int TEST_MODE_START_REPL = 2;
 // Test mode: Delete a logical replication slot.
        int TEST_MODE_DROP_SLOT = 3;
        // Enable different test modes.
        int testMode = TEST_MODE_START_REPL;

        try {
            Class.forName(driver);
        } catch (Exception e) {
            e.printStackTrace();
            return;
        }

        try {
            Properties properties = new Properties();
            PGProperty.USER.set(properties, "user");
            PGProperty.PASSWORD.set(properties, "passwd");
    // For logical replication, the following three attributes are mandatory.
            PGProperty.ASSUME_MIN_SERVER_VERSION.set(properties, "9.4");
            PGProperty.REPLICATION.set(properties, "database");
            PGProperty.PREFER_QUERY_MODE.set(properties, "simple");
            conn = (PgConnection) DriverManager.getConnection(sourceURL, properties);
            System.out.println("connection success!");

            if(testMode == TEST_MODE_CREATE_SLOT){
                conn.getReplicationAPI()
                        .createReplicationSlot()
                        .logical()
                        .withSlotName("replication_slot") // If the character string contains uppercase letters, the
uppercase letters are automatically converted to lowercase letters.
                        .withOutputPlugin("mppdb_decoding")
                        .make();
            }else if(testMode == TEST_MODE_START_REPL) {
                // Create a replication slot before enabling this mode.
                LogSequenceNumber waitLSN = LogSequenceNumber.valueOf("6F/E3C53568");
                PGReplicationStream stream = conn
                        .getReplicationAPI()
                        .replicationStream()
                        .logical()
                        .withSlotName("replication_slot")
                        .withSlotOption("include-xids", false)
                        .withSlotOption("skip-empty-xacts", true)
                        .withStartPosition(waitLSN)
                        .withSlotOption("parallel-decode-num", 10) // Decoding thread concurrency
                        .withSlotOption("white-table-list", "public.t1,public.t2") // Whitelist
                        .withSlotOption("standby-connection", true) // Forcible standby decoding
                        .withSlotOption("decode-style", "t") // Decoding format
                        .withSlotOption("sending-batch", 1) // Sending decoding results in batches
                        .withSlotOption("max-txn-in-memory", 100) // The memory threshold for flushing a single
decoding transaction to disks is 100 MB.
                        .withSlotOption("max-reorderbuffer-in-memory", 50) // The total memory threshold for
flushing decoding transactions that are being handled to disks is 50 GB.
                        .withSlotOption("exclude-users", 'userA') // The logical log of the transaction executed by
user A is not returned.
                        .withSlotOption("include-user", true) // The BEGIN logical log of the transaction contains
the user name.
                        .start();
                while (true) {
                    ByteBuffer byteBuffer = stream.readPending();

                    if (byteBuffer == null) {
                        TimeUnit.MILLISECONDS.sleep(10L);
                        continue;
                    }

                    int offset = byteBuffer.arrayOffset();
                    byte[] source = byteBuffer.array();
                    int length = source.length - offset;
                    System.out.println(new String(source, offset, length));

                    // If the LSN needs to be flushed, call the following APIs based on the service requirements:
                    //LogSequenceNumber lastRecv = stream.getLastReceiveLSN();
                    //stream.setFlushedLSN(lastRecv);
                    //stream.forceUpdateStatus();

                }
            }else if(testMode == TEST_MODE_DROP_SLOT){
                conn.getReplicationAPI()
                        .dropReplicationSlot("replication_slot");
            }
        } catch (Exception e) {
            e.printStackTrace();
            return;
        }
      }
}
```

5.3.16 Example: Parameters for Connecting to the Database in Different Scenarios

NOTE

In the following example, **host:port** represents a node, where **host** indicates the name or IP address of the server where the database resides, and **port** indicates the port number of the server where the database resides.

DR

A customer has two database instances. Database instance A is the production database instance, and database instance B is the DR database instance. When the customer performs a DR switchover, database instance A is demoted to the DR database instance, and database instance B is promoted the production database instance. In this case, to avoid application restart or re-release caused by modifications on the configuration file, the customer can write database instances A and B to the connection string when initializing the configuration file. If the primary database instance cannot be connected, the driver attempts to connect to the DR database instance. For example, database instance A consists of *node1*, *node2*, and *node3*, and database instance B consists of *node4*, *node5*, and *node6*.

The URL can be configured as follows:

```
jdbc:postgresql://node1,node2,node3,node4,node5,node6/database?priorityServers=3
```

If you want to connect to both the primary cluster and hosts in the primary cluster, set **targetServerType** to **master**. The URL can be configured as follows:

```
jdbc:postgresql://node1,node2,node3,node4,node5,node6/database?
priorityServers=3&targetServerType=master
```

Load Balancing

A customer has a centralized database instance that consists of one primary node and two standby nodes, that is, *node1*, *node2*, and *node3*. *node1* is the primary node, and *node2* and *node3* are the standby nodes.

If the customer wants to evenly distribute the connections established on the same application to three nodes, the URL can be configured as follows:

```
jdbc:postgresql://node1,node2,node3/database?loadBalanceHosts=true
```

CAUTION

When **loadBalanceHosts** is used, if the connection is established on the standby DN, write operations cannot be performed. If read and write operations are required, do not set this parameter.

Automatic Selection of the Primary Node

A customer has a centralized database instance that consists of one primary node and two standby nodes, that is, *node1*, *node2*, and *node3*. *node1* is the primary node, and *node2* and *node3* are the standby nodes.

If the customer requires that the application connection be established on the primary DN and a new primary node be automatically selected to establish the connection during the primary/standby switchover, configure the URL as follows:

```
jdbc:postgresql://node1,node2,node3/database?targetServerType=master
```

Log Diagnosis Scenario

If a customer encounters slow data import or some errors that are difficult to analyze, the trace log function can be enabled for diagnosis. The URL can be configured as follows:

```
jdbc:postgresql://node1/database?loggerLevel=trace&loggerFile=jdbc.log
```

High Performance

A customer may execute the same SQL statement for multiple times with different input parameters. To improve the execution efficiency, the **prepareThreshold** parameter can be enabled to avoid repeatedly generating execution plans. The URL can be configured as follows:

```
jdbc:postgresql://node1/database?prepareThreshold=5
```

A customer queries 10 million data records at a time. To prevent memory overflow caused by simultaneous return of the data records, the **defaultRowFetchSize** parameter can be used. The URL can be configured as follows:

```
jdbc:postgresql://node1/database?defaultRowFetchSize=50000
```

A customer needs to insert 10 million data records in batches. To improve efficiency, the **batchMode** parameter can be used. The URL can be configured as follows:

```
jdbc:postgresql://node1/database?batchMode=true
```

Case Conversion

In the Oracle database, metadata is stored in uppercase letters by default. In the GaussDB, metadata is stored in lowercase letters by default. Therefore, after the metadata is migrated from Oracle to GaussDB, the uppercase letters changes to lowercase letters. If the original service involves the processing of uppercase metadata, you can enable this parameter. However, you are advised to modify the service code instead of using this method to solve the problem. If you have to use this function, ensure that the metadata in the current database is in lowercase to avoid problems.

```
jdbc:postgresql://node1/database?uppercaseAttributeName=true
```

The APIs involved in DatabaseMetaData can be directly invoked based on input parameters. The methods of using the APIs involved in ResultSetMetaData are as follows:

```
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery("select * from test_supper");
ResultSetMetaData rsmd = rs.getMetaData();
for (int i = 1; i <= rsmd.getColumnCount(); i++) {
   System.out.println(rsmd.getColumnLabel(i) + "   " + rsmd.getColumnName(i));
}
```

5.3.17 JDBC API Reference

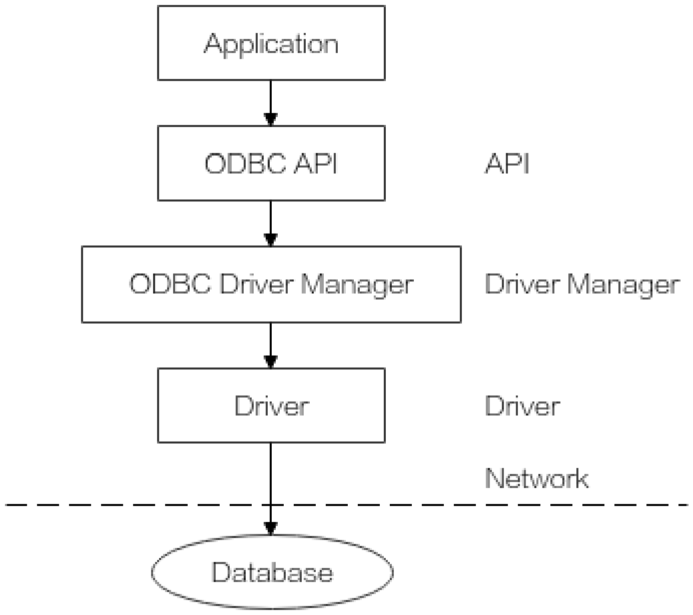
For details, see **JDBC**.

5.4 Development Based on ODBC

Open Database Connectivity (ODBC) is a Microsoft API for accessing databases based on the X/OPEN CLI. Applications interact with the database through the APIs provided by ODBC, which enhances their portability, scalability, and maintainability.

Figure 5-2 shows the system structure of ODBC.

Figure 5-2 ODBC system structure



GaussDB supports ODBC in the following environments.

Table 5-9 OSs Supported by ODBC

OS	Platform
CentOS 6.4/6.5/6.6/6.7/6.8/6.9/7.0/7.1/7.2/7.3/7.4	x86_64
CentOS 7.6	ARM64
EulerOS 2.0 SP2/SP3	x86_64
EulerOS 2.0 SP8	ARM64
Kylin V10	x86_64
Kylin V10	ARM64

The ODBC Driver Manager running on UNIX or Linux can be unixODBC or iODBC. unixODBC-2.3.7 is used as the component for connecting to the database.

Windows has a native ODBC Driver Manager. You can locate **Data Sources (ODBC)** by choosing **Control Panel** > **Administrative Tools**.

NOTE

The current database ODBC driver is based on an open-source version and may be incompatible with data types tinyint, smalldatetime, nvarchar, and nvarchar2.

5.4.1 ODBC Packages, Dependent Libraries, and Header Files

ODBC Packages for the Linux OS

Obtain **release package** provided by GaussDB.

Obtain the **GaussDB-Kernel_VxxxRxxxCxx.x-xxxxx-64bit-Odbc.tar.gz** package from the release package. In the Linux OS, header files (including **sql.h** and **sqlext.h**) and a library file (**libodbc.so**) are required in application development. The header files and library file can be obtained from the **unixODBC-2.3.0** installation package.

ODBC Packages for the Windows OS

Obtain **release package** provided by GaussDB.

Obtain the package from the release package. The ODBC package name is **GaussDB-Kernel_VxxxRxxxCxx.x-Windows-Odbc-X86.tar.gz**. In the Windows OS, the required header files and library files are system-resident.

5.4.2 Configuring a Data Source in the Linux OS

The ODBC DRIVER (**psqlodbcw.so**) provided by GaussDB can be used after it has been configured in a data source. To configure a data source, you must configure the **odbc.ini** and **odbcinst.ini** files on the server. The two files are generated during the unixODBC compilation and installation, and are saved in the **/usr/local/etc** directory by default.

Procedure

Step 1 Obtain the source code package of unixODBC by following link:

https://sourceforge.net/projects/unixodbc/files/unixODBC

After the download, validate the integrity based on the integrity validation algorithm provided by the community.

Step 2 Install unixODBC. It does not matter if unixODBC of another version has been installed.

Currently, unixODBC-2.2.1 is not supported. For example, to install unixODBC-2.3.0, run the commands below. unixODBC is installed in the **/usr/local** directory by default. The data source file is generated in the **/usr/local/etc** directory, and the library file is generated in the **/usr/local/lib** directory.

```
tar zxvf unixODBC-2.3.0.tar.gz
cd unixODBC-2.3.0
# Modify the configure file. (If it does not exist, modify the configure.ac file.) Find LIB_VERSION.
# Change the value of LIB_VERSION to 1:0:0 to compile a *.so.1 dynamic library with the same dependency on psqlodbcw.so.
vim configure

./configure --enable-gui=no #To perform compilation on an ARM server, add the configure parameter --build=aarch64-unknown-linux-gnu.
make
# The installation may require root permissions.
make install
```

Step 3 Replace the GaussDB client driver.

Decompress the **GaussDB-Kernel_VxxxRxxxCxx.x-xxxxx-64bit-Odbc.tar.gz** package. After the decompression, the **lib** and **odbc** folders are generated. The **odbc** folder contains another **lib** folder. Copy all dynamic libraries in the **/lib** and **/odbc/lib** folders to the **/usr/local/lib** directory.

Step 4 Configure a data source.

1. Configure the ODBC driver file.

   Add the following content to the **/usr/local/etc/odbcinst.ini** file:

```
[GaussMPP]
Driver64=/usr/local/lib/psqlodbcw.so
setup=/usr/local/lib/psqlodbcw.so
```

   For descriptions of the parameters in the **odbcinst.ini** file, see **Table 5-10**.

Table 5-10 odbcinst.ini configuration parameters

Parameter	Description	Example
[DriverName]	Driver name, corresponding to Driver in DSN.	[DRIVER_N]
Driver64	Path of the dynamic driver library.	Driver64=/usr/local/lib/psqlodbcw.so
setup	Driver installation path, which is the same as the dynamic library path in Driver64.	setup=/usr/local/lib/psqlodbcw.so

2. Configure the data source file.

   Add the following content to the **/usr/local/etc/odbc.ini** file:

```
[MPPODBC]
Driver=GaussMPP
Servername=10.145.130.26 (IP address of the server where the database resides)
Database=postgres  (database name)
Username=omm  (database username)
Password=    (user password of the database)
Port=8000 (listening port of the database)
Sslmode = allow
```

   For descriptions of the parameters in the **odbc.ini** file, see **Table 5-11**.

Table 5-11 odbc.ini configuration parameters

Parameter	Description	Example
[DSN]	Data source name.	[MPPODBC]
Driver	Driver name, corresponding to DriverName in **odbcinst.ini**.	Driver = DRIVER_N

Parameter	Description	Example
Servername	Server IP address. Multiple IP addresses can be configured.	Servername = 10.145.130.26
Database	Name of the database to connect to.	Database = postgres
Username	Database username.	Username = omm
Password	Database user password.	Password= **NOTE** After a user established a connection, the ODBC driver automatically clears their password stored in memory. However, if this parameter is configured, UnixODBC will cache data source files, which may cause the password to be stored in the memory for a long time. When you connect to an application, you are advised to send your password through an API instead of writing it in a data source configuration file. After the connection has been established, immediately clear the memory segment where your password is stored.
Port	Port number of the server.	Port = 8000
Sslmode	Whether to enable SSL.	Sslmode = allow
Debug	If this parameter is set to **1**, the **mylog** file of the PostgreSQL ODBC driver will be printed. The directory generated for storing logs is **/tmp/**. If this parameter is set to **0**, no directory is generated.	Debug = 1

Parameter	Description	Example
UseServerSidePrepare	Whether to enable the extended query protocol for the database. The value can be **0** or **1**. The default value is **1**, indicating that the extended query protocol is enabled.	UseServerSidePrepare = 1
UseBatchProtocol	Whether to enable the batch query protocol. If it is enabled, DML performance can be improved. The value can be **0** or **1**. The default value is **1**. If this parameter is set to **0**, the batch query protocol is disabled (mainly for communication with earlier database versions). If this parameter is set to **1** and **support_batch_bind** is set to **on**, the batch query protocol is enabled.	UseBatchProtocol = 1
ForExtensionConnec-tor	This parameter specifies whether the savepoint is sent.	ForExtensionConnector = 1

Parameter	Description	Example
UnamedPrepStmtThreshold	Each time **SQLFreeHandle** is invoked to release statements, ODBC sends a **Deallocate plan_name** statement to the server. A large number of such statements exist in the service. To reduce the number of the statements to be sent, **stmt->plan_name** is left empty so that the database can identify them as unnamed statements. This parameter is added to control the threshold for unnamed statements.	UnamedPrepStmtThreshold = 100
ConnectionExtraInfo	Whether to display the driver deployment path and process owner in the **connection_info** parameter mentioned in **connection_info**.	ConnectionExtraInfo = 1 **NOTE** The default value is **0**. If this parameter is set to **1**, the ODBC driver reports the driver deployment path and process owner to the database and displays the information in the **connection_info** parameter (see **connection_info**). In this case, you can query the information from **PG_STAT_ACTIVITY**.
BoolAsChar	If this parameter is set to **Yes**, the Boolean value is mapped to the SQL_CHAR type. If this parameter is not set, the value is mapped to the SQL_BIT type.	BoolsAsChar = Yes
RowVersioning	When an attempt is made to update a row of data, setting this parameter to **Yes** allows the application to detect whether the data has been modified by other users.	RowVersioning = Yes

Parameter	Description	Example
ShowSystemTables	By default, the driver regards the system catalog as a common SQL table.	ShowSystemTables = Yes

The valid values of **Sslmode** are as follows:

Table 5-12 Sslmode options

Sslmode	Whether SSL Encryption Is Enabled	Description
disable	No	SSL connection is not enabled.
allow	Possible	If the database server requires SSL connection, SSL connection can be enabled. However, authenticity of the database server will not be verified.
prefer	Possible	If the database supports SSL connection, SSL connection is recommended. However, authenticity of the database server will not be verified.
require	Yes	SSL connection is required and data is encrypted. However, authenticity of the database server will not be verified.
verify-ca	Yes	SSL connection is required and whether the database has a trusted certificate will be verified.
verify-full	Yes	SSL connection is required. In addition to the check scope specified by **verify-ca**, the system checks whether the name of the host where the database resides is the same as that on the certificate. GaussDB does not support this mode.

Step 5 Configure the environment variables on the client.

```
vim ~/.bashrc
```

Add the following information to the configuration file:

```
export LD_LIBRARY_PATH=/usr/local/lib/:$LD_LIBRARY_PATH
export ODBCSYSINI=/usr/local/etc
export ODBCINI=/usr/local/etc/odbc.ini
```

Step 6 Run the following command to validate the addition:

```
source ~/.bashrc
```

----End

Verifying the Data Source Configuration

Run the **./isql -v** *MPPODBC* command (***MPPODBC*** is the data source name).

- If the following information is displayed, the configuration is correct and the connection succeeds.

```
+---------------------------------------+
| Connected!                            |
|                                       |
| sql-statement                         |
| help [tablename]                      |
| quit                                  |
|                                       |
+---------------------------------------+
SQL>
```

- If error information is displayed, the configuration is incorrect. Check the configuration.

Troubleshooting

- [UnixODBC]connect to server failed: no such file or directory

  Possible causes:

  – An incorrect or unreachable database IP address or port was configured.

    Check the **Servername** and **Port** configuration items in data sources.

  – Server monitoring is improper.

    If **Servername** and **Port** are correctly configured, ensure the proper NIC and port are monitored based on the database server configurations in the procedure in this section.

  – Firewall and network gatekeeper settings are improper.

    Check firewall settings, and ensure that the database communication port is trusted.

    Check to ensure network gatekeeper settings are proper (if any).

- [unixODBC]The password-stored method is not supported.

  Possible causes:

  The **sslmode** configuration item is not configured in the data sources.

  Solution:

  Set the configuration item to **allow** or a higher level. For details, see **Table 5-12**.

- Server common name "xxxx" does not match host name "xxxxx"

  Possible causes:

  When **verify-full** is used for SSL encryption, the driver checks whether the host name in certificates is the same as the actual one.

  Solution:

  To solve this problem, use **verify-ca** to stop checking host names, or generate a set of CA certificates containing the actual host names.

- Driver's SQLAllocHandle on SQL_HANDLE_DBC failed

  Possible causes:

  The executable file (such as the **isql** tool of unixODBC) and the database driver (**psqlodbcw.so**) depend on different library versions of ODBC, such as

libodbc.so.1 and **libodbc.so.2**. You can verify this problem by using the following method:

```
ldd `which isql` | grep odbc
ldd psqlodbcw.so | grep odbc
```

If the suffix digits of the outputs **libodbc.so** are different or indicate different physical disk files, this problem exists. Both **isql** and **psqlodbcw.so** load **libodbc.so**. If different physical files are loaded, different ODBC libraries with the same function list conflict with each other in a visible domain. As a result, the database driver cannot be loaded.

Solution:

Uninstall the unnecessary unixODBC, such as libodbc.so.2, and create a soft link with the same name and the .so.2 suffix for the remaining libodbc.so.1 library.

- [unixODBC][Driver Manager]Invalid attribute value

  The unixODBC version may not be the recommended one. You are advised to run the **odbcinst --version** command to check the unixODBC version in the environment.

- authentication method 10 not supported.

  If this error occurs on an open-source client, the cause may be:

  The database stores only the SHA-256 hash of the password, but the open-source client supports only MD5 hashes.

  NOTE

  - The database stores the hashes of user passwords instead of actual passwords.
  - If a password is updated or a user is created, both types of hashes will be stored, compatible with open-source authentication protocols.
  - An MD5 hash can only be generated using the original password, but the password cannot be obtained by reversing its SHA-256 hash. Passwords in the old version will only have SHA-256 hashes and not support MD5 authentication.
  - The MD5 encryption algorithm has lower security and poses security risks. Therefore, you are advised to use a more secure encryption algorithm.

  To solve this problem, you can update the user password (see **ALTER USER**) or create a user (see **CREATE USER**) having the same permissions as the faulty user.

- unsupported frontend protocol 3.51: server supports 1.0 to 3.0

  The database version is too early or the database is an open-source database. Use the driver of the required version to connect to the database.

- isql: error while loading shared libraries: xxx

  The dynamic library does not exist in the environment. You need to install the corresponding library.

5.4.3 Configuring a Data Source in the Windows OS

Configure an ODBC data source using the ODBC data source manager preinstalled in the Windows OS.

Procedure

Step 1 Replace the GaussDB client driver.

Decompress **GaussDB-Kernel-VxxxRxxxCxx-Windows-Odbc.tar.gz** and double-click **psqlodbc.exe** (32-bit) to install the driver.

Step 2 Open Driver Manager.

Use the ODBC Driver Manager for the 32-bit OS to configure the data source. (Currently, only the ODBC Driver Manager for the 32-bit OS is supported. The following description assumes that the OS is installed on drive C. If the OS is installed on another drive, change the path accordingly.)

- For a 64-bit OS, open **C:\Windows\SysWOW64\odbcad32.exe**. Do not choose **Control Panel** > **Administrative Tools** > **Data Sources (ODBC)**.

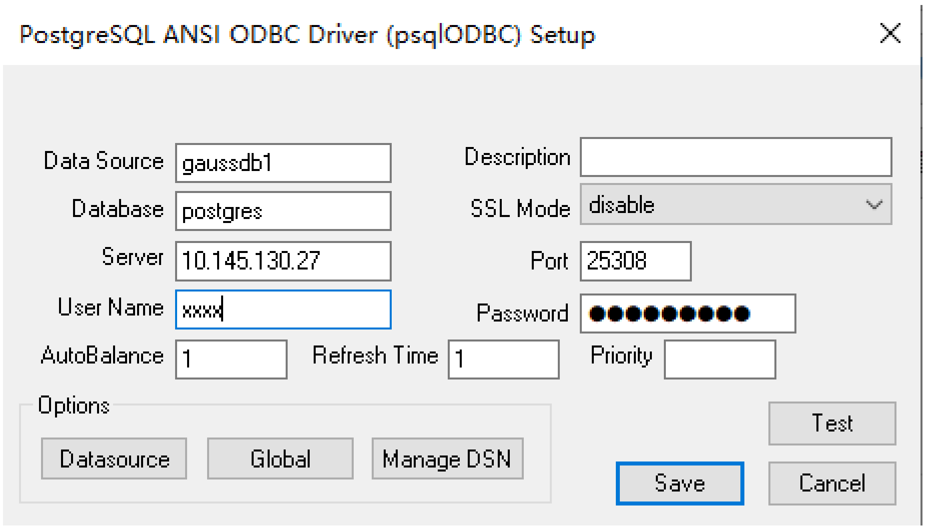
  NOTE

  WoW64 is short for Windows 32-bit on Windows 64-bit. **C:\Windows\SysWOW64\** stores the 32-bit environment on a 64-bit OS. **C:\Windows\System32\** stores the environment consistent with the current OS. For technical details, see Windows technical documents.

- For a 32-bit OS, open **C:\Windows\System32\odbcad32.exe** or choose **Computer** > **Control Panel** > **Administrative Tools** > **Data Sources (ODBC)** to open Driver Manager.

Step 3 Configure a data source.

On the **User DSN** tab, click **Add** and choose **PostgreSQL Unicode** for setup.


PostgreSQL ANSI ODBC Driver (psqlODBC) Setup

Data Source	gaussdb1	Description	
Database	postgres	SSL Mode	disable
Server	10.145.130.27	Port	25308
User Name	xxxx	Password	●●●●●●●●●

AutoBalance 1 Refresh Time 1 Priority

NOTICE

The entered username and password will be recorded in the Windows registry and you do not need to enter them again when connecting to the database next time. For security purposes, you are advised to delete sensitive information before clicking **Save** and enter the required username and password again when using ODBC APIs to connect to the database.

Step 4 Enable the SSL mode.

Copy the **client.crt**, **client.key**, **client.key.cipher**, and **client.key.rand** files to the manually created directory **%APPDATA%\postgresql**. Change **client** in the file names to **postgres**, for example, change **client.key** to **postgres.key**. Copy the **cacert.pem** file to the **%APPDATA%\postgresql** directory and change its name to **root.crt**.

NOTE

%APPDATA% is located in **C:\Users\[username]\AppData** by default, and its values is specified by customers during installation.

Change the value of **SSL Mode** in Step 2 to **require**.

Table 5-13 sslmode options

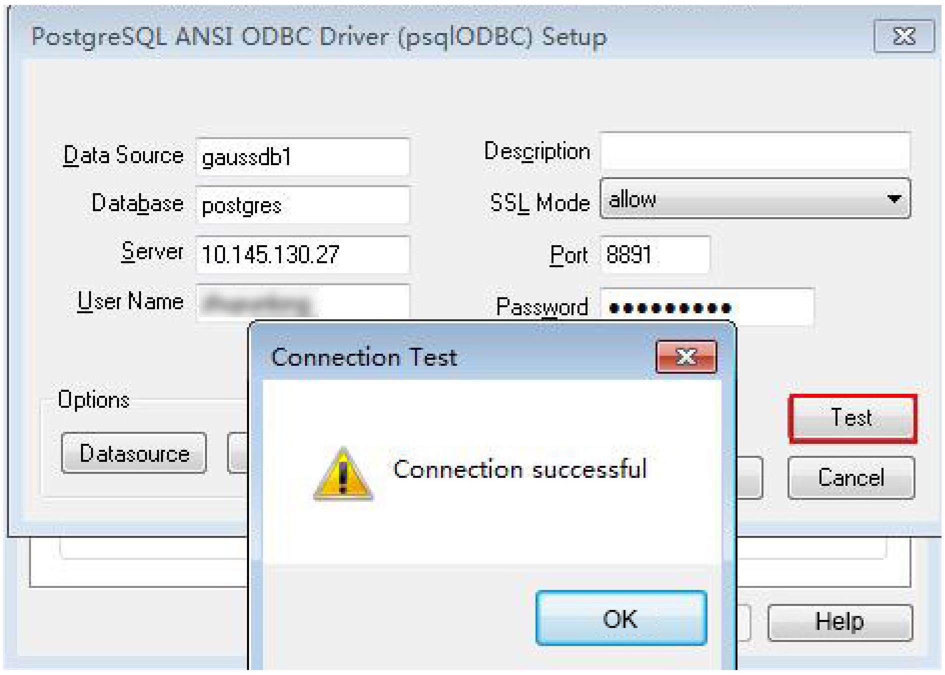
sslmode	Whether SSL Encryption Is Enabled	Description
disable	No	SSL connection is not enabled.
allow	Possible	If the database server requires SSL connection, SSL connection can be enabled. However, authenticity of the database server will not be verified.
prefer	Possible	If the database supports SSL connection, SSL connection is preferred. However, authenticity of the database server will not be verified.
require	Yes	SSL connection is required and data is encrypted. However, authenticity of the database server will not be verified.
verify-ca	Yes	SSL connection is required and whether the database has a trusted certificate will be verified. Currently, Windows ODBC does not support cert authentication.
verify-full	Yes	SSL connection is required. In addition to the check scope specified by **verify-ca**, the system checks whether the name of the host where the database resides is the same as that on the certificate. Currently, Windows ODBC does not support cert authentication.

----End

Verifying the Data Source Configuration

Click **Test**.

- If the following information is displayed, the configuration is correct and the connection succeeds.



PostgreSQL ANSI ODBC Driver (psqlODBC) Setup

Data Source: gaussdb1
Database: postgres
Server: 10.145.130.27
User Name: [redacted]
Description:
SSL Mode: allow
Port: 8891
Password: ●●●●●●●●●●

Connection Test

Connection successful

- If error information is displayed, the configuration is incorrect. Check the configuration.

Troubleshooting

- connect to server failed: no such file or directory

  Possible causes:

  – An incorrect or unreachable database IP address or port was configured.

    Check the **Servername** and **Port** configuration items in data sources.

  – Server monitoring is improper.

    If **Servername** and **Port** are correctly configured, ensure the proper NIC and port are monitored based on the database server configurations in the procedure in this section.

  – Firewall and network gatekeeper settings are improper.

    Check firewall settings, and ensure that the database communication port is trusted.

    Check to ensure network gatekeeper settings are proper (if any).

- The password-stored method is not supported.

  Possible causes:

  sslmode is not configured for the data source. Set this configuration item to **allow** or a higher level to enable SSL connection. For details about **sslmode**, see **Table 5-13**.

- authentication method 10 not supported.

  If this error occurs on an open-source client, the cause may be:

  The database stores only the SHA-256 hash of the password, but the open-source client supports only MD5 hashes.

NOTE

- The database stores the hashes of user passwords instead of actual passwords.
- If a password is updated or a user is created, both types of hashes will be stored, compatible with open-source authentication protocols.
- An MD5 hash can only be generated using the original password, but the password cannot be obtained by reversing its SHA-256 hash. Passwords in the old version will only have SHA-256 hashes and not support MD5 authentication.
- The MD5 encryption algorithm has lower security and poses security risks. Therefore, you are advised to use a more secure encryption algorithm.

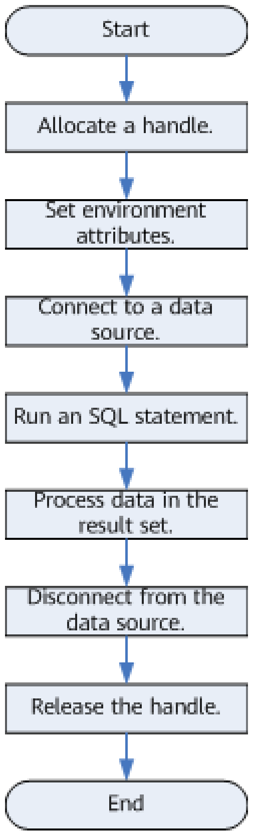
To solve this problem, you can update the user password (see **ALTER USER**) or create a user (see **CREATE USER**) having the same permissions as the faulty user.

- unsupported frontend protocol 3.51: server supports 1.0 to 3.0

  The database version is too early or the database is an open-source database. Use the driver of the required version to connect to the database.

5.4.4 Development Process

Figure 5-3 ODBC-based application development process


Start
Allocate a handle.
Set environment attributes.
Connect to a data source.
Run an SQL statement.
Process data in the result set.
Disconnect from the data source.
Release the handle.
End

APIs Involved in the Development Process

Table 5-14 API description

Function	API
Allocate a handle	**SQLAllocHandle** is a generic function for allocating a handle. It can replace the following functions: ● **SQLAllocEnv**: allocate an environment handle ● **SQLAllocConnect**: allocate a connection handle ● **SQLAllocStmt**: allocate a statement handle
Set environment attributes	**SQLSetEnvAttr**
Set connection attributes	**SQLSetConnectAttr**
Set statement attributes	**SQLSetStmtAttr**
Connect to a data source	**SQLConnect**
Bind a buffer to a column in the result set	**SQLBindCol**
Bind the parameter marker of an SQL statement to a buffer	**SQLBindParameter**
Return the error message of the last operation	**SQLGetDiagRec**
Prepare an SQL statement for execution	**SQLPrepare**
Run a prepared SQL statement	**SQLExecute**
Run an SQL statement directly	**SQLExecDirect**
Fetch the next row (or rows) from the result set	**SQLFetch**
Return data in a column of the result set	**SQLGetData**
Get the column information from a result set	**SQLColAttribute**
Disconnect from a data source	**SQLDisconnect**

Function	API
Release a handle	**SQLFreeHandle** is a generic function for releasing a handle. It can replace the following functions: ● **SQLFreeEnv**: release an environment handle ● **SQLFreeConnect**: release a connection handle ● **SQLFreeStmt**: release a statement handle

NOTE

If an execution request (not in a transaction block) received in the database contains multiple statements, the request is packed into a transaction. If one of the statements fails, the entire request will be rolled back.

WARNING

ODBC connects applications to the database and delivers the SQL commands sent by an application to the database. It does not parse the SQL syntax. Therefore, when confidential information (such as a plaintext password) is written into the SQL statement delivered by an application, the confidential information is exposed in the driver log.

5.4.5 Example: Common Functions and Batch Binding

Code for Common Functions

```
//The following example shows how to obtain data from GaussDB through the ODBC interface.
// DBtest.c (compile with: libodbc.so)
#include <stdlib.h>
#include <stdio.h>
#include <sqlext.h>
#ifdef WIN32
#include <windows.h>
#endif
SQLHENV       V_OD_Env;        // Handle ODBC environment
SQLHSTMT      V_OD_hstmt;      // Handle statement
SQLHDBC       V_OD_hdbc;       // Handle connection
char          typename[100];
SQLINTEGER    value = 100;
SQLINTEGER    V_OD_erg,V_OD_buffer,V_OD_err,V_OD_id;
int main(int argc,char *argv[])
{
     // 1. Allocate an environment handle.
     V_OD_erg = SQLAllocHandle(SQL_HANDLE_ENV,SQL_NULL_HANDLE,&V_OD_Env);
     if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
     {
          printf("Error AllocHandle\n");
          exit(0);
     }
     // 2. Set environment attributes (version information).
     SQLSetEnvAttr(V_OD_Env, SQL_ATTR_ODBC_VERSION, (void*)SQL_OV_ODBC3, 0);
     // 3. Allocate a connection handle.
     V_OD_erg = SQLAllocHandle(SQL_HANDLE_DBC, V_OD_Env, &V_OD_hdbc);
     if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
     {
          SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
          exit(0);
      }
      // 4. Set connection attributes.
      SQLSetConnectAttr(V_OD_hdbc, SQL_ATTR_AUTOCOMMIT, SQL_AUTOCOMMIT_ON, 0);
     // 5. Connect to the data source. userName and password indicate the username and password for
connecting to the database. Set them as needed.
     // If the username and password have been set in the odbc.ini file, you do not need to set userName or
password here, retaining "" for them. However, you are not advised to do so because the username and
password will be disclosed if the permission for odbc.ini is abused.
      V_OD_erg = SQLConnect(V_OD_hdbc, (SQLCHAR*) "gaussdb", SQL_NTS,
                         (SQLCHAR*) "userName", SQL_NTS,  (SQLCHAR*) "password", SQL_NTS);
      if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
      {
          printf("Error SQLConnect %d\n",V_OD_erg);
          SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
          exit(0);
      }
      printf("Connected !\n");
      // 6. Set statement attributes.
      SQLSetStmtAttr(V_OD_hstmt,SQL_ATTR_QUERY_TIMEOUT,(SQLPOINTER *)3,0);
       // 7. Allocate a statement handle.
      SQLAllocHandle(SQL_HANDLE_STMT, V_OD_hdbc, &V_OD_hstmt);
      // 8. Run SQL statements.
      SQLExecDirect(V_OD_hstmt,"drop table IF EXISTS customer_t1",SQL_NTS);
      SQLExecDirect(V_OD_hstmt,"CREATE TABLE customer_t1(c_customer_sk INTEGER, c_customer_name
VARCHAR(32));",SQL_NTS);
      SQLExecDirect(V_OD_hstmt,"insert into customer_t1 values(25,li)",SQL_NTS);
      // 9. Prepare for execution.
      SQLPrepare(V_OD_hstmt,"insert into customer_t1 values(?)",SQL_NTS);
      // 10. Bind parameters.
      SQLBindParameter(V_OD_hstmt,1,SQL_PARAM_INPUT,SQL_C_SLONG,SQL_INTEGER,0,0,
                     &value,0,NULL);
      // 11. Run prepared statements.
      SQLExecute(V_OD_hstmt);
      SQLExecDirect(V_OD_hstmt,"select id from testtable",SQL_NTS);
      // 12. Obtain attributes of a specific column in the result set.
      SQLColAttribute(V_OD_hstmt,1,SQL_DESC_TYPE,typename,100,NULL,NULL);
      printf("SQLColAtrribute %s\n",typename);
      // 13. Bind the result set.
      SQLBindCol(V_OD_hstmt,1,SQL_C_SLONG, (SQLPOINTER)&V_OD_buffer,150,
              (SQLLEN *)&V_OD_err);
      // 14. Obtain data in the result set by executing SQLFetch.
      V_OD_erg=SQLFetch(V_OD_hstmt);
      // 15. Obtain and return data by executing SQLGetData.
      while(V_OD_erg != SQL_NO_DATA)
      {
          SQLGetData(V_OD_hstmt,1,SQL_C_SLONG,(SQLPOINTER)&V_OD_id,0,NULL);
          printf("SQLGetData ----ID = %d\n",V_OD_id);
          V_OD_erg=SQLFetch(V_OD_hstmt);
      };
      printf("Done !\n");
      // 16. Disconnect data source connections and release handles.
      SQLFreeHandle(SQL_HANDLE_STMT,V_OD_hstmt);
      SQLDisconnect(V_OD_hdbc);
      SQLFreeHandle(SQL_HANDLE_DBC,V_OD_hdbc);
      SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
      return(0);
 }
```

Code for Batch Processing

```
/**********************************************************************
* Enable UseBatchProtocol in the data source and set the database parameter support_batch_bind
* to on.
* The CHECK_ERROR command is used to check and print error information.
* This example is used to interactively obtain the DSN, data volume to be processed, and volume of ignored
data from users, and insert required data into the test_odbc_batch_insert table.
***********************************************************************/
```

```
#include <stdio.h>
#include <stdlib.h>
#include <sql.h>
#include <sqlext.h>
#include <string.h>

void Exec(SQLHDBC hdbc, SQLCHAR* sql)
{
    SQLRETURN retcode;                  // Return status
    SQLHSTMT hstmt = SQL_NULL_HSTMT;    // Statement handle
    SQLCHAR     loginfo[2048];

    // Allocate Statement Handle
    retcode = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLAllocHandle(SQL_HANDLE_STMT) failed");
        return;
    }

    // Prepare Statement
    retcode = SQLPrepare(hstmt, (SQLCHAR*) sql, SQL_NTS);
    sprintf((char*)loginfo, "SQLPrepare log: %s", (char*)sql);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLPrepare(hstmt, (SQLCHAR*) sql, SQL_NTS) failed");
        return;
    }

    // Execute Statement
    retcode = SQLExecute(hstmt);
    sprintf((char*)loginfo, "SQLExecute stmt log: %s", (char*)sql);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLExecute(hstmt) failed");
        return;
    }
    // Free Handle
    retcode = SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
    sprintf((char*)loginfo, "SQLFreeHandle stmt log: %s", (char*)sql);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLFreeHandle(SQL_HANDLE_STMT, hstmt) failed");
        return;
    }
}

int main ()
{
    SQLHENV  henv  = SQL_NULL_HENV;
    SQLHDBC  hdbc  = SQL_NULL_HDBC;
    int      batchCount = 1000; // Amount of data that is bound in batches
    SQLLEN   rowsCount = 0;
    int      ignoreCount = 0; // Amount of data that is not imported to the database among the data that is
bound in batches

    SQLRETURN   retcode;
    SQLCHAR     dsn[1024] = {'\0'};
    SQLCHAR     loginfo[2048];

    do
    {
        if (ignoreCount > batchCount)
        {
            printf("ignoreCount(%d) should be less than batchCount(%d)\n", ignoreCount, batchCount);
        }
    }while(ignoreCount > batchCount);

    retcode = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
```

```
if (!SQL_SUCCEEDED(retcode)) {
    printf("SQLAllocHandle failed");
    goto exit;
}

// Set ODBC Verion
retcode = SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION,
                                    (SQLPOINTER*)SQL_OV_ODBC3, 0);

if (!SQL_SUCCEEDED(retcode)) {
    printf("SQLSetEnvAttr failed");
    goto exit;
}

// Allocate Connection
retcode = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);

if (!SQL_SUCCEEDED(retcode)) {
    printf("SQLAllocHandle failed");
    goto exit;
}


// Set Login Timeout
retcode = SQLSetConnectAttr(hdbc, SQL_LOGIN_TIMEOUT, (SQLPOINTER)5, 0);

if (!SQL_SUCCEEDED(retcode)) {
    printf("SQLSetConnectAttr failed");
    goto exit;
}


// Set Auto Commit
retcode = SQLSetConnectAttr(hdbc, SQL_ATTR_AUTOCOMMIT,
                                    (SQLPOINTER)(1), 0);

if (!SQL_SUCCEEDED(retcode)) {
    printf("SQLSetConnectAttr failed");
    goto exit;
}


// Connect to DSN
// gaussdb indicates the name of the data source used by users.
sprintf(loginfo, "SQLConnect(DSN:%s)", dsn);
retcode = SQLConnect(hdbc, (SQLCHAR*) "gaussdb", SQL_NTS,
                           (SQLCHAR*) NULL, 0, NULL, 0);

if (!SQL_SUCCEEDED(retcode)) {
    printf("SQLConnect failed");
    goto exit;
}

// init table info.
Exec(hdbc, "drop table if exists test_odbc_batch_insert");
Exec(hdbc, "create table test_odbc_batch_insert(id int primary key, col varchar2(50))");

// The following code constructs the data to be inserted based on the data volume entered by users:
{
    SQLRETURN retcode;
    SQLHSTMT hstmtinesrt = SQL_NULL_HSTMT;
    int          i;
    SQLCHAR      *sql = NULL;
    SQLINTEGER   *ids  = NULL;
    SQLCHAR      *cols = NULL;
    SQLLEN       *bufLenIds = NULL;
    SQLLEN       *bufLenCols = NULL;
    SQLUSMALLINT *operptr = NULL;
```

```
        SQLUSMALLINT *statusptr = NULL;
        SQLULEN      process = 0;

        // Data is constructed by column. Each column is stored continuously.
        ids = (SQLINTEGER*)malloc(sizeof(ids[0]) * batchCount);
        cols = (SQLCHAR*)malloc(sizeof(cols[0]) * batchCount * 50);
       // Data size in each row for a column
        bufLenIds = (SQLLEN*)malloc(sizeof(bufLenIds[0]) * batchCount);
        bufLenCols = (SQLLEN*)malloc(sizeof(bufLenCols[0]) * batchCount);
       // Whether this row needs to be processed. The value is SQL_PARAM_IGNORE or
SQL_PARAM_PROCEED.
        operptr = (SQLUSMALLINT*)malloc(sizeof(operptr[0]) * batchCount);
        memset(operptr, 0, sizeof(operptr[0]) * batchCount);
       // Processing result of the row
       // Note: In the database, a statement belongs to one transaction. Therefore, data is processed as a
unit. Either all data is inserted successfully or all data fails to be inserted.
        statusptr = (SQLUSMALLINT*)malloc(sizeof(statusptr[0]) * batchCount);
        memset(statusptr, 88, sizeof(statusptr[0]) * batchCount);

        if (NULL == ids || NULL == cols || NULL == bufLenCols || NULL == bufLenIds)
        {
            fprintf(stderr, "FAILED:\tmalloc data memory failed\n");
            goto exit;
        }

        for (int i = 0; i < batchCount; i++)
        {
            ids[i] = i;
            sprintf(cols + 50 * i, "column test value %d", i);
            bufLenIds[i] = sizeof(ids[i]);
            bufLenCols[i] = strlen(cols + 50 * i);
            operptr[i] = (i < ignoreCount) ? SQL_PARAM_IGNORE : SQL_PARAM_PROCEED;
        }

        // Allocate Statement Handle
        retcode = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmtinesrt);

        if (!SQL_SUCCEEDED(retcode)) {
            printf("SQLAllocHandle failed");
            goto exit;
        }

        // Prepare Statement
        sql = (SQLCHAR*)"insert into test_odbc_batch_insert values(?, ?)";
        retcode = SQLPrepare(hstmtinesrt, (SQLCHAR*) sql, SQL_NTS);
        sprintf((char*)loginfo, "SQLPrepare log: %s", (char*)sql);

        if (!SQL_SUCCEEDED(retcode)) {
            printf("SQLPrepare failed");
            goto exit;
        }

        retcode = SQLSetStmtAttr(hstmtinesrt, SQL_ATTR_PARAMSET_SIZE, (SQLPOINTER)batchCount,
sizeof(batchCount));

        if (!SQL_SUCCEEDED(retcode)) {
            printf("SQLSetStmtAttr failed");
            goto exit;
        }

        retcode = SQLBindParameter(hstmtinesrt, 1, SQL_PARAM_INPUT, SQL_C_SLONG, SQL_INTEGER,
sizeof(ids[0]), 0,&(ids[0]), 0, bufLenIds);

        if (!SQL_SUCCEEDED(retcode)) {
            printf("SQLBindParameter failed");
            goto exit;
        }

        retcode = SQLBindParameter(hstmtinesrt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, 50, 50,
cols, 50, bufLenCols);

        if (!SQL_SUCCEEDED(retcode)) {
            printf("SQLBindParameter failed");
            goto exit;
        }

        retcode = SQLSetStmtAttr(hstmtinesrt, SQL_ATTR_PARAMS_PROCESSED_PTR, (SQLPOINTER)&process,
sizeof(process));

        if (!SQL_SUCCEEDED(retcode)) {
            printf("SQLSetStmtAttr failed");
            goto exit;
        }

        retcode = SQLSetStmtAttr(hstmtinesrt, SQL_ATTR_PARAM_STATUS_PTR, (SQLPOINTER)statusptr,
sizeof(statusptr[0]) * batchCount);

        if (!SQL_SUCCEEDED(retcode)) {
            printf("SQLSetStmtAttr failed");
            goto exit;
        }

        retcode = SQLSetStmtAttr(hstmtinesrt, SQL_ATTR_PARAM_OPERATION_PTR, (SQLPOINTER)operptr,
sizeof(operptr[0]) * batchCount);

        if (!SQL_SUCCEEDED(retcode)) {
            printf("SQLSetStmtAttr failed");
            goto exit;
        }

        retcode = SQLExecute(hstmtinesrt);
        sprintf((char*)loginfo, "SQLExecute stmt log: %s", (char*)sql);

        if (!SQL_SUCCEEDED(retcode)) {
            printf("SQLExecute(hstmtinesrt) failed");
            goto exit;

        retcode = SQLRowCount(hstmtinesrt, &rowsCount);

        if (!SQL_SUCCEEDED(retcode)) {
            printf("SQLRowCount failed");
            goto exit;
        }

        if (rowsCount != (batchCount - ignoreCount))
        {
            sprintf(loginfo, "(batchCount - ignoreCount)(%d) != rowsCount(%d)", (batchCount - ignoreCount),
rowsCount);

            if (!SQL_SUCCEEDED(retcode)) {
                printf("SQLExecute failed");
                goto exit;
            }
        }
        else
        {
            sprintf(loginfo, "(batchCount - ignoreCount)(%d) == rowsCount(%d)", (batchCount - ignoreCount),
rowsCount);

            if (!SQL_SUCCEEDED(retcode)) {
                printf("SQLExecute failed");
                goto exit;
            }
        }

        // check row number returned
        if (rowsCount != process)
        {
            sprintf(loginfo, "process(%d) != rowsCount(%d)", process, rowsCount);

            if (!SQL_SUCCEEDED(retcode)) {
                printf("SQLExecute failed");
                goto exit;
            }
        }
        else
        {
            sprintf(loginfo, "process(%d) == rowsCount(%d)", process, rowsCount);

            if (!SQL_SUCCEEDED(retcode)) {
                printf("SQLExecute failed");
                goto exit;
            }
        }

        for (int i = 0; i < batchCount; i++)
        {
            if (i < ignoreCount)
            {
                if (statusptr[i] != SQL_PARAM_UNUSED)
                {
                    sprintf(loginfo, "statusptr[%d](%d) != SQL_PARAM_UNUSED", i, statusptr[i]);

                    if (!SQL_SUCCEEDED(retcode)) {
                        printf("SQLExecute failed");
                        goto exit;
                    }
                }
            }
            else if (statusptr[i] != SQL_PARAM_SUCCESS)
            {
                sprintf(loginfo, "statusptr[%d](%d) != SQL_PARAM_SUCCESS", i, statusptr[i]);

                if (!SQL_SUCCEEDED(retcode)) {
                    printf("SQLExecute failed");
                    goto exit;
                }
            }
        }

        retcode = SQLFreeHandle(SQL_HANDLE_STMT, hstmtinesrt);
        sprintf((char*)loginfo, "SQLFreeHandle hstmtinesrt");

        if (!SQL_SUCCEEDED(retcode)) {
            printf("SQLFreeHandle failed");
            goto exit;
        }
    }


exit:
    (void) printf ("\nComplete.\n");

    // Connection
    if (hdbc != SQL_NULL_HDBC) {
        SQLDisconnect(hdbc);
        SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
    }

    // Environment
    if (henv != SQL_NULL_HENV)
        SQLFreeHandle(SQL_HANDLE_ENV, henv);

    return 0;
}
```

5.4.6 Typical Application Scenarios and Configurations

Log Diagnosis Scenario

ODBC logs are classified into unixODBC driver manager logs and psqlODBC driver logs. The former is used to trace whether the application API is successfully executed, and the latter is used to locate problems based on DFX logs generated during underlying implementation.

The unixODBC log needs to be configured in the **odbcinst.ini** file:

```
[ODBC]
Trace=Yes
TraceFile=/path/to/odbctrace.log

[GaussMPP]
Driver64=/usr/local/lib/psqlodbcw.so
setup=/usr/local/lib/psqlodbcw.so
```

You only need to add the following information to the **odbc.ini** file:

```
[gaussdb]
Driver=GaussMPP
Servername=10.10.0.13 (database server IP address)
...
Debug=1 (Enable the debug log function of the driver.)
```

NOTE

The unixODBC logs are generated in the path configured by **TraceFile**. The psqlODBC generates the **mylog_*xxx*.log** file in the **/tmp/** directory.

High Performance

If a large amount of data needs to be inserted, you are advised to perform the following operations:

- You need to set **UseBatchProtocol** to **1** in the **odbc.ini** file and **support_batch_bind** to **on** in the database.
- The ODBC program binding type must be the same as that in the database.
- The character set of the client is the same as that of the database.
- The transaction is committed manually.

odbc.ini configuration file:

```
[gaussdb]
Driver=GaussMPP
Servername=10.10.0.13 (database server IP address)
...
UseBatchProtocol=1 (enabled by default)
ConnSettings=set client_encoding=UTF8 (Set the character code on the client to be the same as that on the server.)
```

Binding type case:

```
#include <stdio.h>
#include <stdlib.h>
#include <sql.h>
#include <sqlext.h>
#include <string.h>
#include <sys/time.h>

#define MESSAGE_BUFFER_LEN  128
```

```
SQLHANDLE   h_env = NULL;
SQLHANDLE   h_conn = NULL;
SQLHANDLE   h_stmt = NULL;
void print_error()
{
    SQLCHAR     Sqlstate[SQL_SQLSTATE_SIZE+1];
    SQLINTEGER  NativeError;
    SQLCHAR     MessageText[MESSAGE_BUFFER_LEN];
    SQLSMALLINT TextLength;
    SQLRETURN   ret = SQL_ERROR;

    ret = SQLGetDiagRec(SQL_HANDLE_STMT, h_stmt, 1, Sqlstate, &NativeError, MessageText, 
MESSAGE_BUFFER_LEN, &TextLength);
    if ( SQL_SUCCESS == ret)
    {
        printf("\n STMT ERROR-%05d %s", NativeError, MessageText);
        return;
    }

    ret = SQLGetDiagRec(SQL_HANDLE_DBC, h_conn, 1, Sqlstate, &NativeError, MessageText, 
MESSAGE_BUFFER_LEN, &TextLength);
    if ( SQL_SUCCESS == ret)
    {
        printf("\n CONN ERROR-%05d %s", NativeError, MessageText);
        return;
    }

    ret = SQLGetDiagRec(SQL_HANDLE_ENV, h_env, 1, Sqlstate, &NativeError, MessageText, 
MESSAGE_BUFFER_LEN, &TextLength);
    if ( SQL_SUCCESS == ret)
    {
        printf("\n ENV ERROR-%05d %s", NativeError, MessageText);
        return;
    }

    return;
}

/* Expect the function to return SQL_SUCCESS. */
#define RETURN_IF_NOT_SUCCESS(func) \
{\
    SQLRETURN ret_value = (func);\
    if (SQL_SUCCESS != ret_value)\
    {\
        print_error();\
        printf("\n failed line = %u: expect SQL_SUCCESS, but ret = %d", __LINE__, ret_value);\
        return SQL_ERROR; \
    }\
}

/* Expect the function to return SQL_SUCCESS. */
#define RETURN_IF_NOT_SUCCESS_I(i, func) \
{\
    SQLRETURN ret_value = (func);\
    if (SQL_SUCCESS != ret_value)\
    {\
        print_error();\
        printf("\n failed line = %u (i=%d): : expect SQL_SUCCESS, but ret = %d", __LINE__, (i), ret_value);\
        return SQL_ERROR; \
    }\
}

/* Expect the function to return SQL_SUCCESS_WITH_INFO. */
#define RETURN_IF_NOT_SUCCESS_INFO(func) \
{\
    SQLRETURN ret_value = (func);\
    if (SQL_SUCCESS_WITH_INFO != ret_value)\
    {\
        print_error();\
        printf("\n failed line = %u: expect SQL_SUCCESS_WITH_INFO, but ret = %d", __LINE__, ret_value);\
        return SQL_ERROR; \
    }\
}

/* Expect the values are the same. */
#define RETURN_IF_NOT(expect, value) \
if ((expect) != (value))\
{\
    printf("\n failed line = %u: expect = %u, but value = %u", __LINE__, (expect), (value)); \
    return SQL_ERROR;\
}

/* Expect the character strings are the same. */
#define RETURN_IF_NOT_STRCMP_I(i, expect, value) \
if (( NULL == (expect) ) || (NULL == (value)))\
{\
    printf("\n failed line = %u (i=%u): input NULL pointer !", __LINE__, (i)); \
    return SQL_ERROR; \
}\
else if (0 != strcmp((expect), (value)))\
{\
    printf("\n failed line = %u (i=%u): expect = %s, but value = %s", __LINE__, (i), (expect), (value)); \
    return SQL_ERROR;\
}


// prepare + execute SQL statement
int execute_cmd(SQLCHAR *sql)
{
    if ( NULL == sql )
    {
        return SQL_ERROR;
    }

    if ( SQL_SUCCESS != SQLPrepare(h_stmt, sql, SQL_NTS))
    {
        return SQL_ERROR;
    }

    if ( SQL_SUCCESS != SQLExecute(h_stmt))
    {
        return SQL_ERROR;
    }

    return SQL_SUCCESS;
}
// execute + commit handle
int commit_exec()
{
    if ( SQL_SUCCESS != SQLExecute(h_stmt))
    {
        return SQL_ERROR;
    }

     // Manual committing
    if ( SQL_SUCCESS != SQLEndTran(SQL_HANDLE_DBC, h_conn, SQL_COMMIT))
    {
        return SQL_ERROR;
    }

    return SQL_SUCCESS;
}

int begin_unit_test()
{
    SQLINTEGER    ret;

    /* Allocate an environment handle. */
    ret = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &h_env);
    if ((SQL_SUCCESS != ret) && (SQL_SUCCESS_WITH_INFO != ret))
    {
        printf("\n begin_unit_test::SQLAllocHandle SQL_HANDLE_ENV failed ! ret = %d", ret);
        return SQL_ERROR;
    }

    /* Set the version number before connection. */
    if (SQL_SUCCESS != SQLSetEnvAttr(h_env, SQL_ATTR_ODBC_VERSION, (SQLPOINTER)SQL_OV_ODBC3, 
0))
    {
        print_error();
        printf("\n begin_unit_test::SQLSetEnvAttr SQL_ATTR_ODBC_VERSION failed ! ret = %d", ret);
        SQLFreeHandle(SQL_HANDLE_ENV, h_env);
        return SQL_ERROR;
    }

    /* Allocate a connection handle. */
    ret = SQLAllocHandle(SQL_HANDLE_DBC, h_env, &h_conn);
    if (SQL_SUCCESS != ret)
    {
        print_error();
        printf("\n begin_unit_test::SQLAllocHandle SQL_HANDLE_DBC failed ! ret = %d", ret);
        SQLFreeHandle(SQL_HANDLE_ENV, h_env);
        return SQL_ERROR;
    }

    /* Establish a connection. */
    ret = SQLConnect(h_conn, (SQLCHAR*) "gaussdb", SQL_NTS,
                            (SQLCHAR*) NULL, 0, NULL, 0);
    if (SQL_SUCCESS != ret)
    {
        print_error();
        printf("\n begin_unit_test::SQLConnect  failed ! ret = %d", ret);
        SQLFreeHandle(SQL_HANDLE_DBC, h_conn);
        SQLFreeHandle(SQL_HANDLE_ENV, h_env);
        return SQL_ERROR;
    }

    /* Allocate a statement handle. */
    ret = SQLAllocHandle(SQL_HANDLE_STMT, h_conn, &h_stmt);
    if (SQL_SUCCESS != ret)
    {
        print_error();
        printf("\n begin_unit_test::SQLAllocHandle SQL_HANDLE_STMT failed ! ret = %d", ret);
        SQLFreeHandle(SQL_HANDLE_DBC, h_conn);
        SQLFreeHandle(SQL_HANDLE_ENV, h_env);
        return SQL_ERROR;
    }

    return SQL_SUCCESS;
}

void end_unit_test()
{
    /* Release a statement handle. */
    if (NULL != h_stmt)
    {
        SQLFreeHandle(SQL_HANDLE_STMT, h_stmt);
    }

    /* Release a connection handle. */
    if (NULL != h_conn)
    {
        SQLDisconnect(h_conn);
        SQLFreeHandle(SQL_HANDLE_DBC, h_conn);
    }

    /* Release an environment handle. */
    if (NULL != h_env)
    {
        SQLFreeHandle(SQL_HANDLE_ENV, h_env);
    }

    return;
}

int main()
{
    // begin test
    if (begin_unit_test() != SQL_SUCCESS)
    {
        printf("\n begin_test_unit failed.");
        return SQL_ERROR;
    }
    // The handle configuration is the same as that in the preceding case
    int          i = 0;
    SQLCHAR*    sql_drop   = "drop table if exists test_bindnumber_001";
    SQLCHAR*    sql_create = "create table test_bindnumber_001("
                             "f4 number, f5 number(10, 2)"
                             ")";
    SQLCHAR*    sql_insert = "insert into test_bindnumber_001 values(?, ?)";
    SQLCHAR*    sql_select = "select * from test_bindnumber_001";
    SQLLEN      RowCount;
    SQL_NUMERIC_STRUCT st_number;
    SQLCHAR     getValue[2][MESSAGE_BUFFER_LEN];

    /* Step 1. Create a table. */
    RETURN_IF_NOT_SUCCESS(execute_cmd(sql_drop));
    RETURN_IF_NOT_SUCCESS(execute_cmd(sql_create));

    /* Step 2.1 Bind parameters using the SQL_NUMERIC_STRUCT structure. */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));

    // First line: 1234.5678
    memset(st_number.val, 0, SQL_MAX_NUMERIC_LEN);
    st_number.precision = 8;
    st_number.scale = 4;
    st_number.sign = 1;
    st_number.val[0] = 0x4E;
    st_number.val[1] = 0x61;
    st_number.val[2] = 0xBC;

    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_NUMERIC,
SQL_NUMERIC, sizeof(SQL_NUMERIC_STRUCT), 4, &st_number, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_NUMERIC,
SQL_NUMERIC, sizeof(SQL_NUMERIC_STRUCT), 4, &st_number, 0, NULL));

    // Disable the automatic commit function.
    SQLSetConnectAttr(h_conn, SQL_ATTR_AUTOCOMMIT, (SQLPOINTER)SQL_AUTOCOMMIT_OFF, 0);

    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    // Second line: 12345678
    memset(st_number.val, 0, SQL_MAX_NUMERIC_LEN);
    st_number.precision = 8;
    st_number.scale = 0;
    st_number.sign = 1;
    st_number.val[0] = 0x4E;
    st_number.val[1] = 0x61;
    st_number.val[2] = 0xBC;

    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_NUMERIC,
SQL_NUMERIC, sizeof(SQL_NUMERIC_STRUCT), 0, &st_number, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_NUMERIC,
SQL_NUMERIC, sizeof(SQL_NUMERIC_STRUCT), 0, &st_number, 0, NULL));
```

```
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    // Third line: 12345678
    memset(st_number.val, 0, SQL_MAX_NUMERIC_LEN);
    st_number.precision = 0;
    st_number.scale = 4;
    st_number.sign = 1;
    st_number.val[0] = 0x4E;
    st_number.val[1] = 0x61;
    st_number.val[2] = 0xBC;

    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_NUMERIC,
SQL_NUMERIC, sizeof(SQL_NUMERIC_STRUCT), 4, &st_number, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_NUMERIC,
SQL_NUMERIC, sizeof(SQL_NUMERIC_STRUCT), 4, &st_number, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);


    /* Step 2.2 Bind parameters by using the SQL_C_CHAR character string in the fourth line. */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    SQLCHAR*    szNumber = "1234.5678";
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_NUMERIC, strlen(szNumber), 0, szNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_NUMERIC, strlen(szNumber), 0, szNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    /* Step 2.3 Bind parameters by using SQL_C_FLOAT in the fifth line. */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    SQLREAL    fNumber = 1234.5678;
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_FLOAT,
SQL_NUMERIC, sizeof(fNumber), 4, &fNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_FLOAT,
SQL_NUMERIC, sizeof(fNumber), 4, &fNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    /* Step 2.4 Bind parameters by using SQL_C_DOUBLE in the sixth line. */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    SQLDOUBLE   dNumber = 1234.5678;
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_DOUBLE,
SQL_NUMERIC, sizeof(dNumber), 4, &dNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_DOUBLE,
SQL_NUMERIC, sizeof(dNumber), 4, &dNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    SQLBIGINT   bNumber1 = 0xFFFFFFFFFFFFFFFF;
    SQLBIGINT   bNumber2 = 12345;

    /* Step 2.5 Bind parameters by using SQL_C_SBIGINT in the seventh line. */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_SBIGINT,
SQL_NUMERIC, sizeof(bNumber1), 4, &bNumber1, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_SBIGINT,
SQL_NUMERIC, sizeof(bNumber2), 4, &bNumber2, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    /* Step 2.6 Bind parameters by using SQL_C_UBIGINT in the eighth line. */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_UBIGINT,
SQL_NUMERIC, sizeof(bNumber1), 4, &bNumber1, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_UBIGINT,
SQL_NUMERIC, sizeof(bNumber2), 4, &bNumber2, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    SQLLEN  lNumber1 = 0xFFFFFFFFFFFFFFFF;
    SQLLEN  lNumber2 = 12345;

    /* Step 2.7 Bind parameters by using SQL_C_LONG in the ninth line. */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_LONG,
SQL_NUMERIC, sizeof(lNumber1), 0, &lNumber1, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_LONG,
SQL_NUMERIC, sizeof(lNumber2), 0, &lNumber2, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    /* Step 2.8 Bind parameters by using SQL_C_ULONG in the tenth line. */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_ULONG,
SQL_NUMERIC, sizeof(lNumber1), 0, &lNumber1, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_ULONG,
SQL_NUMERIC, sizeof(lNumber2), 0, &lNumber2, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    SQLSMALLINT sNumber = 0xFFFF;

    /* Step 2.9 Bind parameters by using SQL_C_SHORT in the eleventh line. */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_SHORT,
SQL_NUMERIC, sizeof(sNumber), 0, &sNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_SHORT,
SQL_NUMERIC, sizeof(sNumber), 0, &sNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    /* Step 2.10 Bind parameters by using SQL_C_USHORT in the twelfth line. */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_USHORT,
SQL_NUMERIC, sizeof(sNumber), 0, &sNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_USHORT,
SQL_NUMERIC, sizeof(sNumber), 0, &sNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    SQLCHAR cNumber = 0xFF;

    /* Step 2.11 Bind parameters by using SQL_C_TINYINT in the thirteenth line. */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_TINYINT,
SQL_NUMERIC, sizeof(cNumber), 0, &cNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_TINYINT,
SQL_NUMERIC, sizeof(cNumber), 0, &cNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    /* Step 2.12 Bind parameters by using SQL_C_UTINYINT in the fourteenth line.*/
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_UTINYINT,
SQL_NUMERIC, sizeof(cNumber), 0, &cNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_UTINYINT,
SQL_NUMERIC, sizeof(cNumber), 0, &cNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    /* Use the character string type to unify the expectation. */
    SQLCHAR*    expectValue[14][2] = {{"1234.5678",             "1234.57"},
                                     {"12345678",              "12345678"},
                                     {"0",                     "0"},
                                     {"1234.5678",             "1234.57"},
                                     {"1234.5677",             "1234.57"},
                                     {"1234.5678",             "1234.57"},
                                     {"-1",                    "12345"},
                                     {"18446744073709551615",  "12345"},
                                     {"-1",                    "12345"},
                                     {"4294967295",            "12345"},
                                     {"-1",                    "-1"},
                                     {"65535",                 "65535"},
                                     {"-1",                    "-1"},
                                     {"255",                   "255"},
                                     };

    RETURN_IF_NOT_SUCCESS(execute_cmd(sql_select));
    while ( SQL_NO_DATA != SQLFetch(h_stmt))
    {
        RETURN_IF_NOT_SUCCESS_I(i, SQLGetData(h_stmt, 1, SQL_C_CHAR, &getValue[0],
MESSAGE_BUFFER_LEN, NULL));
        RETURN_IF_NOT_SUCCESS_I(i, SQLGetData(h_stmt, 2, SQL_C_CHAR, &getValue[1],
MESSAGE_BUFFER_LEN, NULL));

        //RETURN_IF_NOT_STRCMP_I(i, expectValue[i][0], getValue[0]);
        //RETURN_IF_NOT_STRCMP_I(i, expectValue[i][1], getValue[1]);
        i++;
    }

    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(i, RowCount);
    SQLCloseCursor(h_stmt);
    /* Final step. Delete the table and restore the environment. */
    RETURN_IF_NOT_SUCCESS(execute_cmd(sql_drop));

    end_unit_test();
}
```

NOTE

In the preceding example, the number column is defined. When the **SQLBindParameter** API is called, the performance of binding SQL_NUMERIC is higher than that of SQL_LONG. If char is used, the data type needs to be converted when data is inserted to the database server, causing a performance bottleneck.

Automatic primary/standby switchover

Example Scenario

If a database instance is configured with one primary DN and multiple standby DNs, write the IP addresses of all DNs into the configuration file. ODBC automatically searches for the primary DN and establishes a connection with it. When a primary/standby switchover occurs, ODBC can also connect to the new primary DN.

5.4.7 ODBC Interface Reference

For details, see **ODBC**.

5.5 Development Based on libpq

libpq is a C application programming interface to GaussDB. **libpq** contains a set of library functions that allow client programs to send query requests to the GaussDB servers and obtain query results. It is also the underlying engine of other GaussDB application interfaces, such as ODBC. This section provides two examples to show how to write code using **libpq**.

5.5.1 Dependent Header Files of libpq

Client programs that use **libpq** must include the header file **libpq-fe.h** and must link with the libpq library.

5.5.2 Development Process

Obtain **release package** provided by GaussDB.

To compile and connect to a libpq source program, perform the following operations:

1. Decompress of **GaussDB-Kernel-VxxxRxxxCxx-xxxxx-64bit-Libpq.tar.gz**. The required header file is stored in the **include** folder, and the **lib** folder contains the required libpq library file.

   NOTE

   In addition to **libpq-fe.h**, the **include** folder contains the header files **postgres_ext.h**, **gs_thread.h**, and **gs_threadlocal.h** by default. These three header files are the dependency files of **libpq-fe.h**.
2. Include the **libpq-fe.h** header file.
   ```
   #include <libpq-fe.h>
   ```
3. Provide the **-I** *directory* option to provide the installation location of the header files. (Sometimes the compiler looks for the default directory, so this option can be ignored.) Example:
   ```
   gcc -I (Directory where the header files are located) -L (Directory where the libpq library is located)
   testprog.c -lpq
   ```
4. If the makefile is used, add the following options to variables *CPPFLAGS*, *LDFLAGS*, and *LIBS*:
   ```
   CPPFLAGS += -I (Directory where the header files are located)
   LDFLAGS += -L (Directory where the libpq library is located)
   LIBS += -lpq
   ```

5.5.3 Example

Code for Common Functions

Example 1:

```
/*
 * testlibpq.c
 */
#include <stdio.h>
#include <stdlib.h>
#include <libpq-fe.h>

static void
```

```
exit_nicely(PGconn *conn)
{
    PQfinish(conn);
    exit(1);
}

int
main(int argc, char **argv)
{
    const char *conninfo;
    PGconn     *conn;
    PGresult   *res;
    int         nFields;
    int         i,j;

    /*
     * This value is used when the user provides the value of the conninfo character string in the command
line.
     * Otherwise, the environment variables or the default values
     * are used for all other connection parameters.
     */
    if (argc > 1)
        conninfo = argv[1];
    else
        conninfo = "dbname=postgres port=42121 host='10.44.133.171' application_name=test
connect_timeout=5 sslmode=allow user='test' password='test_1234'";

    /* Connect to the database. */
    conn = PQconnectdb(conninfo);

    /* Check whether the backend connection has been successfully established. */
    if (PQstatus(conn) != CONNECTION_OK)
    {
        fprintf(stderr, "Connection to database failed: %s",
                PQerrorMessage(conn));
        exit_nicely(conn);
    }

    /*
     * Since a cursor is used in the test case, a transaction block is required.
     * Put all data in one "select * from pg_database"
     * PQexec() is too simple and is not recommended.
     */

    /* Start a transaction block. */
    res = PQexec(conn, "BEGIN");
    if (PQresultStatus(res) != PGRES_COMMAND_OK)
    {
        fprintf(stderr, "BEGIN command failed: %s", PQerrorMessage(conn));
        PQclear(res);
        exit_nicely(conn);
    }

    /*
     * PQclear PGresult should be executed when it is no longer needed, to avoid memory leakage.
     */
    PQclear(res);

    /*
     * Fetch data from the pg_database system catalog.
     */
    res = PQexec(conn, "DECLARE myportal CURSOR FOR select * from pg_database");
    if (PQresultStatus(res) != PGRES_COMMAND_OK)
    {
        fprintf(stderr, "DECLARE CURSOR failed: %s", PQerrorMessage(conn));
        PQclear(res);
        exit_nicely(conn);
    }
    PQclear(res);
    res = PQexec(conn, "FETCH ALL in myportal");
    if (PQresultStatus(res) != PGRES_TUPLES_OK)
    {
        fprintf(stderr, "FETCH ALL failed: %s", PQerrorMessage(conn));
        PQclear(res);
        exit_nicely(conn);
    }

    /* First, print out the attribute name. */
    nFields = PQnfields(res);
    for (i = 0; i < nFields; i++)
        printf("%-15s", PQfname(res, i));
    printf("\n\n");

    /* Print lines. */
    for (i = 0; i < PQntuples(res); i++)
    {
        for (j = 0; j < nFields; j++)
            printf("%-15s", PQgetvalue(res, i, j));
        printf("\n");
    }

    PQclear(res);

    /* Close the portal. We do not need to check for errors. */
    res = PQexec(conn, "CLOSE myportal");
    PQclear(res);

    /* End the transaction. */
    res = PQexec(conn, "END");
    PQclear(res);

    /* Close the database connection and clean up the database. */
    PQfinish(conn);

    return 0;
}
```

Example 2:

```
/*
 * testlibpq2.c
 *      Test out-of-line parameters and binary I/Os.
 *
 * Before running this example, run the following command to populate a database:
 *
 *
 * CREATE TABLE test1 (i int4, t text);
 *
  * INSERT INTO test1 values (2, 'ho there');
 *
 * The expected output is as follows:
 *
 *
 * tuple 0: got
 *  i = (4 bytes) 2
 *  t = (8 bytes) 'ho there'
 *
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <libpq-fe.h>

/* for ntohl/htonl */
#include <netinet/in.h>
#include <arpa/inet.h>
```

```
static void
exit_nicely(PGconn *conn)
{
    PQfinish(conn);
    exit(1);
}

/*
 * This function is used to print out the query results. The results are in binary format
* and fetched from the table created in the comment above.
 */
static void
show_binary_results(PGresult *res)
{
    int         i;
    int         i_fnum,
               t_fnum;

    /* Use PQfnumber to avoid assumptions about field order in the result. */
    i_fnum = PQfnumber(res, "i");
    t_fnum = PQfnumber(res, "t");

    for (i = 0; i < PQntuples(res); i++)
    {
        char       *iptr;
        char       *tptr;
        int         ival;

        /* Obtain the field value. (Ignore the possibility that they may be null). */
        iptr = PQgetvalue(res, i, i_fnum);
        tptr = PQgetvalue(res, i, t_fnum);

        /*
         * The binary representation of INT4 is the network byte order,
         * which is better to be replaced with the local byte order.
         */
        ival = ntohl(*((uint32_t *) iptr));

        /*
         * The binary representation of TEXT is text. Since libpq can append a zero byte to it,
         *  and think of it as a C string.
         *
         */

        printf("tuple %d: got\n", i);
        printf(" i = (%d bytes) %d\n",
               PQgetlength(res, i, i_fnum), ival);
        printf(" t = (%d bytes) '%s'\n",
               PQgetlength(res, i, t_fnum), tptr);
        printf("\n\n");
    }
}

int
main(int argc, char **argv)
{
    const char *conninfo;
    PGconn     *conn;
    PGresult   *res;
    const char *paramValues[1];
    int         paramLengths[1];
    int         paramFormats[1];
    uint32_t    binaryIntVal;

    /*
     * If the user provides a parameter on the command line,
     * The value of this parameter is a conninfo character string. Otherwise,
    * Use environment variables or default values.
     */
    if (argc > 1)
        conninfo = argv[1];
    else
        conninfo = "dbname=postgres port=42121 host='10.44.133.171' application_name=test
connect_timeout=5 sslmode=allow user='test' password='test_1234'";

    /* Connect to the database. */
    conn = PQconnectdb(conninfo);

     /* Check whether the connection to the server was successfully established. */
    if (PQstatus(conn) != CONNECTION_OK)
    {
        fprintf(stderr, "Connection to database failed: %s",
                PQerrorMessage(conn));
        exit_nicely(conn);
    }

    /* Convert the integer value "2" to the network byte order. */
    binaryIntVal = htonl((uint32_t) 2);

    /* Set the parameter array for PQexecParams. */
    paramValues[0] = (char *) &binaryIntVal;
    paramLengths[0] = sizeof(binaryIntVal);
    paramFormats[0] = 1;        /* Binary */

    res = PQexecParams(conn,
                       "SELECT * FROM test1 WHERE i = $1::int4",
                       1,       /* One parameter */
                       NULL,    /* Enable the backend to deduce the parameter type. */
                       paramValues,
                       paramLengths,
                       paramFormats,
                       1);      /* require binary results. */

    if (PQresultStatus(res) != PGRES_TUPLES_OK)
    {
        fprintf(stderr, "SELECT failed: %s", PQerrorMessage(conn));
        PQclear(res);
        exit_nicely(conn);
    }

    show_binary_results(res);

    PQclear(res);

    /* Close the database connection and clean up the database. */
    PQfinish(conn);

    return 0;
}
```

Example 3:

```
/*
 * testlibpq3.c Test PQprepare
 * /
#include <stdio.h>
#include <stdlib.h>
#include <libpq-fe.h>
int main(int argc, char * argv[])
{
    PGconn *conn;
    PGresult * res;
    ConnStatusType pgstatus;
    char connstr[1024];
    char cmd_sql[2048];
    int nParams = 0;
    int paramLengths[5];
    int paramFormats[5];
    Oid paramTypes[5];
    char * paramValues[5];
    int i, cnt;
    char cid[32];
    int k;
    sprintf(connstr,
            "hostaddr=%s dbname=%s port=%d user=%s password=%s",
            "8.92.5.173", "postgres", 5432, "bot", "Gauss_234");
    conn = PQconnectdb(connstr);
    pgstatus = PQstatus(conn);
    if (pgstatus == CONNECTION_OK)
    {
        printf("Connect database success!\n");
    }
    else
    {
        printf("Connect database fail:%s\n",PQerrorMessage(conn));
        return -1;
    }
    sprintf(cmd_sql, "SELECT b FROM t01 WHERE a = $1");
    paramTypes[0] = 23;
    res = PQprepare(conn,
                    "pre_name",
                    cmd_sql,
                    1,
                    paramTypes);
   if( PQresultStatus(res) != PGRES_COMMAND_OK )
    {
        printf("Failed to prepare SQL : %s\n: %s\n",cmd_sql, PQerrorMessage(conn));
        PQfinish(conn);
        return -1;
    }
    PQclear(res);
    paramValues[0] = cid;
    for (k=0; k<2; k++)
    {
        sprintf(cid, "%d", 1);
        paramLengths[0] = 6;
        paramFormats[0] = 0;
        res = PQexecPrepared(conn,
                            "pre_name",
                            1,
                            paramValues,
                            paramLengths,
                            paramFormats,
                            0);
        if( (PQresultStatus(res) != PGRES_COMMAND_OK ) && (PQresultStatus(res) != PGRES_TUPLES_OK))
        {
            printf("%s\n",PQerrorMessage(conn));
            PQclear(res);
            PQfinish(conn);
            return -1;
        }
        cnt = PQntuples(res);
        printf("return %d rows\n", cnt);
        for (i=0; i<cnt; i++)
        {
            printf("row %d: %s\n", i, PQgetvalue(res, i, 0));
        }
        PQclear(res);
    }
    PQfinish(conn);
    return 0;
}
```

5.5.4 libpq API Reference

For details, see **libpq**.

5.5.5 Link Parameters

Table 5-15 Link parameters

Parameter	Description
host	Name of the host to connect to. If the host name starts with a slash (/), Unix-domain socket communications instead of TCP/IP communications are used. The value is the directory where the socket file is stored. If **host** is not specified, the default behavior is to connect to the Unix-domain socket in the **/tmp** directory (or the socket directory specified during database installation). On a machine without a Unix-domain socket, the default behavior is to connect to **localhost**. You can specify multiple host names by using a character string separated by commas (,). Multiple host names can be specified.
hostaddr	IP address of the host to connect to. The value is in standard IPv4 address format, for example, 172.28.40.9. If the machine supports IPv6, IPv6 address can also be used. If a non-null string is specified, TCP/IP communications are used. You can specify multiple IP addresses by using a character string separated by commas (,). Multiple IP addresses can be specified. Replacing **host** with **hostaddr** can prevent applications from querying host names, which may be important for applications with time constraints. However, a host name is required for GSSAPI or SSPI authentication methods. Therefore, the following rules are used: 1. If **host** is specified but **hostaddr** is not, a query for the host name will be executed. 2. If **hostaddr** is specified but **host** is not, the value of **hostaddr** is the server network address. If the host name is required by authentication, the connection attempt fails. 3. If both **host** and **hostaddr** are specified, the value of **hostaddr** is the server network address. The value of **host** is ignored unless it is required by authentication, in which case it is used as the host name. **NOTICE** ● If **host** is not the server name in the network address specified by **hostaddr**, the authentication may fail. ● If neither **host** nor **hostaddr** is specified, libpq will use a local Unix-domain socket for connection. If the machine does not have a Unix-domain socket, it will attempt to connect to **localhost**.
port	Port number of the host server, or the socket file name extension for Unix-domain connections. You can specify multiple port numbers by using a character string separated by commas (,). Multiple port numbers can be specified.

Parameter	Description
user	Name of the user to connect as. By default, the username is the same as the operating system name of the user running the application.
dbname	Database name. The default value is the same as the username.
password	Password to be used if the server requires password authentication.
connect_timeout	Maximum timeout period of the connection, in seconds (in decimal integer string). The value **0** or null indicates infinity. You are not advised to set the connection timeout period to a value less than 2 seconds.
client_encoding	Client encoding for the connection. In addition to the values accepted by the corresponding server options, you can use **auto** to determine the correct encoding from the current environment in the client (the *LC_CTYPE* environment variable in the Unix system).
tty	This parameter can be ignored. (This parameter was used to specify the location to which the debugging output of the server was sent).
options	Adds command-line options to send to the server at run time.
application_name	Current user identity.
fallback_application_name	Specifies a backup value for the **application_name** parameter. This value is used if no value is set for **application_name** through a connection parameter or the *PGAPPNAME* environment variable. It is useful to specify a backup value in a common tool program that wants to set a default application name but does not want it to be overwritten by the user.
keepalives	Whether TCP keepalive is enabled on the client side. The default value is **1**, indicating that the function is enabled. The value **0** indicates that the function is disabled. Ignore this parameter for Unix-domain connections.
keepalives_idle	The number of seconds of inactivity after which TCP should send a keepalive message to the server. The value **0** indicates that the default value is used. Ignore this parameter for Unix-domain connections or if keep-alive is disabled.
keepalives_interval	The number of seconds after which a TCP keepalive message that is not acknowledged by the server should be retransmitted. The value **0** indicates that the default value is used. Ignore this parameter for Unix-domain connections or if keep-alive is disabled.

Parameter	Description
keepalives_count	Controls the number of times that keepalive messages are sent through TCP. The value **0** indicates that the default value is used. Ignore this parameter for Unix-domain connections or if keep-alive is disabled.
tcp_user_timeout	Specifies the maximum duration for which transmitted data can remain unacknowledged before the TCP connection is forcibly closed on an operating system that supports the **TCP_USER_TIMEOUT** socket option. The value **0** indicates that the default value is used. Ignore this parameter for Unix-domain connections.
rw_timeout	Sets the read and write timeout interval of the client connection.
sslmode	Specifies whether to enable SSL encryption. ● **disable**: SSL connection is disabled. ● **allow**: If the database server requires SSL connection, SSL connection can be enabled. However, authenticity of the database server will not be verified. ● **prefer**: If the database supports SSL connection, SSL connection is preferred. However, authenticity of the database server will not be verified. ● **require**: SSL connection is required and data is encrypted. However, authenticity of the database server will not be verified. ● **verify-ca**: SSL connection is required. Currently, Windows ODBC does not support cert authentication. ● **verify-full**: SSL connection is required. Currently, Windows ODBC does not support cert authentication.
sslcompression	If this parameter is set to **1** (default value), the data transmitted over the SSL connection is compressed (this requires that the OpenSSL version 0.9.8 or later). If this parameter is set to **0**, compression will be disabled (this requires OpenSSL 1.0.0 or later). If a connection without SSL is established, this parameter is ignored. If the OpenSSL version in use does not support this parameter, it will also be ignored. Compression takes up CPU time, but it increases throughput when the bottleneck is the network. If CPU performance is a limiting factor, disabling compression can improve response time and throughput.
sslcert	This parameter specifies the file name of the client SSL certificate. It replaces the default **~/.postgresql/postgresql.crt**. If no SSL connection is established, this parameter is ignored.

Parameter	Description
sslkey	This parameter specifies the location of the key used for the client certificate. It can specify the name of a file used to replace the default **~/.postgresql/postgresql.key**, or specify a key obtained from an external "engine" that is a loadable module of OpenSSL. The description of an external engine should consist of a colon-separated engine name and an engine-related key identifier. If no SSL connection is established, this parameter is ignored.
sslrootcert	This parameter specifies the name of a file that contains the SSL Certificate Authority (CA) certificate. If the file exists, the system authenticates the server certificate issued by one of these authorities. The default value is **~/.postgresql/root.crt**.
sslcrl	This parameter specifies the file name of the SSL Certificate Revocation List (CRL). If a certificate listed in this file exists, the server certificate authentication will be rejected. The default value is **~/.postgresql/root.crl**.
requirepeer	This parameter specifies the OS user of the server, for example, **requirepeer=postgres**. When a Unix-domain socket connection is established, if this parameter is set, the client checks whether the server process is running under the specified username at the beginning of the connection. If not, the connection will be interrupted by an error. This parameter can be used to provide server authentication similar to that of the SSL certificate on TCP/IP connections. Note that if the Unix domain socket is in **/tmp** or another public writable location, any user can start a server for listening to the location. Use this parameter to ensure that your device is connected to a server that is run by a trusted user. This option is supported only on platforms that implement the peer authentication method.
krbsrvname	This parameter specifies the Kerberos service name used for GSSAPI authentication. For successful Kerberos authentication, this value must match the service name specified in the server configuration.
gsslib	This parameter specifies the GSS library used for GSSAPI authentication. It is used only in the Windows OS. If this parameter is set to **gssapi**, **libpq** is forced to use the GSSAPI library to replace the default SSPI for authentication.
service	This parameter specifies the name of the service for which the additional parameter is used. It specifies a service name in **pg_service.conf** that holds the additional connection parameters. This allows the application to specify only one service name so that the connection parameters can be centrally maintained.
authtype	**authtype** is no longer used, so it is marked as a parameter not to be displayed. It is retained in an array so as not to reject the **conninfo** string from old applications that might still try to set it.

Parameter	Description
remote_nodename	Specifies the name of the remote node connected to the local node.
localhost	Specifies the local host in a connection channel.
localport	Specifies the local port in a connection channel.
fencedUdfRPCMode	Specifies whether the fenced udf RPC protocol uses UNIX domain sockets or special socket file names. The default value is **0**, indicating that the UNIX domain socket mode is used and the file type is .s.PGSQL.%d. To use the fenced UDF mode, set this parameter to **1**. In this case, the file type is .s.fencedMaster_unixdomain.
replication	Specifies whether the connection should use replication protocols instead of common protocols. Protocols with this parameter configured are internal protocols used for PostgreSQL replication connections and tools such as **pg_basebackup**, while they can also be used by third-party applications. The following values, which are case-insensitive, are supported: ● **true**, **on**, **yes**, and **1**: Specify that the physical replication mode is connected. ● **database**: Specifies that the logical replication mode and the database specified by **dbname** are connected. ● **false**, **off**, **no**, and **0**: Specify that the connection is a regular connection, which is the default behavior. In physical or logical replication mode, only simple query protocols can be used.
backend_version	Specifies the backend version to be passed to the remote end.
prototype	Sets the current protocol level. The default value is **PROTO_TCP**.
enable_ce	Specifies whether a client is allowed to connect to a fully encrypted database. The current feature is a lab feature. Contact Huawei technical support before using it. The default value is **0**. To enable this function, change the value to **1**.
connection_info	The value of **connection_info** is a JSON character string consisting of **driver_name**, **driver_version**, **driver_path**, and **os_user**. If the value is not null, use **connection_info** and ignore **connectionExtraInf**. If the value is null, a connection information string related to **libpq** is generated. When **connectionExtraInf** is set to **false**, the value of **connection_info** consists of only **driver_name** and **driver_version**.

Parameter	Description
connectionExtraInf	Specifies whether the value of **connection_info** contains extension information. The default value is **0**. If the value contains other information, set this parameter to **1**.
target_session_attrs	Specifies the type of the host to be connected. The connection is successful only when the host type is the same as the configured value. The rules for setting **target_session_attrs** are as follows: ● **any** (default value): All types of hosts can be connected. ● **read-write**: The connection is set up only when the connected host is readable and writable. ● **read-only**: Only readable hosts can be connected. ● **primary**: Only the primary server in the primary/standby systems can be connected. ● **standby**: Only the standby server in the primary/standby systems can be connected. ● **prefer-standby**: The system first attempts to find a standby node for connection. If all hosts in the **hosts** list fail to be connected, try the **any** mode.

5.6 Psycopg-based Development

Psycopg is a Python API used to execute SQL statements and provides a unified access API for PostgreSQL and GaussDB. Applications can perform data operations based on Psycopg. Psycopg2 is the encapsulation of libpq and is implemented using the C language, which is efficient and secure. It provides cursors on both clients and servers, asynchronous communication and notification, and the COPY TO and COPY FROM functions. Psycopg2 supports multiple types of Python out-of-the-box and adapts to PostgreSQL data types. Through the flexible object adaptation system, you can extend and customize the adaptation. Psycopg2 is compatible with Unicode and Python 3.

GaussDB supports Psycopg2 features and allows Psycopg2 to be connected in SSL mode.

Table 5-16 Platforms supported by Psycopg

OS	Platform
EulerOS 2.5	x86_64
EulerOS 2.8	ARM64
Kylin	x86_64
Kylin	ARM64

NOTICE

OpenSSL of GaussDB is linked during Psycopg2 compilation. It may be incompatible with OpenSSL of the operating system. If incompatibility occurs, for example, "version 'OPENSSL_1_1_1f' not found" is displayed, use the environment variable **LD_LIBRARY_PATH** to isolate the OpenSSL provided by the operating system and the OpenSSL on which GaussDB depends.

For example, when the application software **client.py** that invokes Psycopg2 is executed, the environment variable is explicitly assigned to the application software.

```
export LD_LIBRARY_PATH=/path/to/gaussdb/libs:$LD_LIBRARY_PATH python client.py
```

In the preceding command, **/path/to/pyscopg2/lib** indicates the directory where the OpenSSL library on which GaussDB depends is located. Change it as required.

5.6.1 Psycopg Package

Obtain **release package** provided by GaussDB.

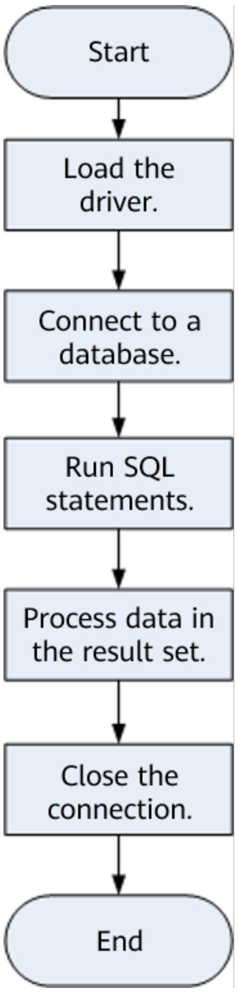
The package is obtained from the release package. The package name is **GaussDB-Kernel_VxxxRxxxCxx.x-***OS version number* **-64bit-Python.tar.gz**.

After the decompression, the following folders are generated:

- **psycopg2**: **psycopg2** library file
- **lib**: **lib** library file

5.6.2 Development Process

Figure 5-4 Application development process based on psycopg2



5.6.3 Loading a Driver

- Before using the driver, perform the following operations:
  a. Decompress the driver package of the corresponding version and copy psycopg2 to the **site-packages** folder in the Python installation directory as the **root** user.
  b. Change the **psycopg2** directory permission to **755**.
  c. Add the **psycopg2** directory to the environment variable *$PYTHONPATH* and validate it.
  d. For non-database users, configure the **lib** directory in *LD_LIBRARY_PATH* after decompression.
- Load a database driver before creating a database connection:

```
import  psycopg2
```

5.6.4 Connecting to the Database

1. Use the **psycopg2.connect** function to obtain the connection object.
2. Use the connection object to create a cursor object.

5.6.5 Executing SQL Statements

1. Construct an operation statement and use **%s** as a placeholder. During execution, psycopg2 will replace the placeholder with the parameter value. You can add the RETURNING clause to obtain the automatically generated column values.
2. The **cursor.execute** method is used to perform operations on one row, and the **cursor.executemany** method is used to perform operations on multiple rows.

5.6.6 Processing the Result Set

1. **cursor.fetchone()**: Fetches the next row in a query result set and returns a sequence. If no data is available, null is returned.
2. **cursor.fetchall()**: Fetches all remaining rows in a query result and returns a list. An empty list is returned when no rows are available.

NOTE

For GaussDB-specific data types, such as tinyint, the corresponding fields in the query result are character strings.

5.6.7 Closing the Connection

After you complete required data operations in a database, close the database connection. Call the close method such as **connection.close()** to close the connection.

CAUTION

This method closes the database connection and does not automatically call **commit()**. If you just close the database connection without calling **commit()** first, changes will be lost.

5.6.8 Connecting to the Database (Using SSL)

When you use psycopy2 to connect to the GaussDB server, you can enable SSL to encrypt the communication between the client and server. To enable SSL, you must have the server certificate, client certificate, and private key files. For details on how to obtain these files, see related documents and commands of OpenSSL.

1. Use the .ini file (the **configparser** package of Python can parse this type of configuration file) to save the configuration information about the database connection.
2. Add SSL connection parameters **sslmode**, **sslcert**, **sslkey**, and **sslrootcert** to the connection options.

   a. **sslmode**: **Table 5-17**
   b. **sslcert**: client certificate path
   c. **sslkey**: client key path
   d. **sslrootcert**: root certificate path
3. Use the **psycopg2.connect** function to obtain the connection object.
4. Use the connection object to create a cursor object.

Table 5-17 sslmode options

sslmode	Whether SSL Encryption Is Enabled	Description
disable	No	SSL connection is not enabled.
allow	Possible	If the database server requires SSL connection, SSL connection can be enabled. However, authenticity of the database server will not be verified.
prefer	Possible	If the database supports SSL connection, SSL connection is preferred. However, authenticity of the database server will not be verified.
require	Yes	SSL connection is required and data is encrypted. However, authenticity of the database server will not be verified.
verify-ca	Yes	The SSL connection must be enabled.
verify-full	Yes	The SSL connection must be enabled. Currently, GaussDB does not support this option.

5.6.9 Example: Common Operations

```
import psycopg2

# Create a connection object.
conn=psycopg2.connect(database="postgres",user="user",password="password",host="localhost",port=port)
cur=conn.cursor() # Create a pointer object.

# Create a connection object (using SSL).
conn = psycopg2.connect(dbname="postgres", user="user", password="password", host="localhost",
port=port,
        sslmode="verify-ca", sslcert="client.crt",sslkey="client.key",sslrootcert="cacert.pem")
Note: If sslcert, sslkey, and sslrootcert are not set, the following files in the .postgresql directory of the
current user are used by default: client.crt,
        client.key, and root.crt.


# Create a table.
cur.execute("CREATE TABLE student(id integer,name varchar,sex varchar);")

# Insert data.
```

```
cur.execute("INSERT INTO student(id,name,sex) VALUES(%s,%s,%s)",(1,'Aspirin','M'))
cur.execute("INSERT INTO student(id,name,sex) VALUES(%s,%s,%s)",(2,'Taxol','F'))
cur.execute("INSERT INTO student(id,name,sex) VALUES(%s,%s,%s)",(3,'Dixheral','M'))

# Obtain the result.
cur.execute('SELECT * FROM student')
results=cur.fetchall()
print (results)

# Close the connection.
conn.commit()
cur.close()
conn.close()

Common connection modes of psycopg2
1. conn = psycopg2.connect(dbname="postgres", user="user", password="password", host="localhost",
port=port)
2. conn = psycopg2.connect("dbname=postgres user=user password=password  host=localhost port=port")
3. Using logs
import logging
import psycopg2
from psycopg2.extras import LoggingConnection

logging.basicConfig(level=logging.DEBUG) # Log level
logger = logging.getLogger(__name__)

db_settings = {
    "user": "user",
    "password": "password",
    "host": "localhost",
    "database": "postgres",
    "port": port
}
conn = psycopg2.connect(connection_factory=LoggingConnection, **db_settings)
conn.initialize(logger)
```

5.6.10 Psycopg API Reference

For details, see **Psycopg**.

5.7 Development Based on the Go Driver

5.7.1 Go Driver Package and Driver Class

Go Driver Package

Obtain **release package** provided by GaussDB.

Obtain the **GaussDB -Kernel-VxxxRxxxCxx-***OS version number***-64bit-Go.tar.gz** package from the release package. Decompress the package to obtain the Go driver source code package.

NOTICE

The Go driver package provided by the database depends on Go 1.13 or later.

Driver Class

When creating a database connection, you need to enter the database driver name **opengauss**.

NOTICE

The Go driver provided by the database does not adapt to mature ORM frameworks (such as XORM) in the industry. As such, the driver name input during database connection creation must be compatible with Postgres and PostgreSQL.

The Go driver of the database cannot coexist with that of PostgreSQL.

5.7.2 Go Code Project Structure

Currently, online import is not supported. You need to place the decompressed Go driver source code package in the local project. The driver code is managed based on go mod. **GO111MODULE** needs to be set to **auto** or **on**. When **go build** or **go run** is executed, the related driver dependency is parsed and downloaded. Currently, **GOPROXY=http://cmc.centralrepo.rnd.huawei.com/go,http://mirrors.tools.huawei.com/goproxy** can be set for Huawei internal users. Set **GOPROXY=https://goproxy.io,direct** for external users.

NOTE

Currently, Huawei's memory dependency repository is under HTTPS reconstruction. In the future, the setting can be changed to **GOPROXY=https://cmc.centralrepo.rnd.huawei.com/go,https://mirrors.tools.huawei.com/goproxy**.

The go mod development project is as follows:

-go

----pkg

----src

-------gitee.com

----------opengauss

-------------openGauss-connector-go-pq

-------Huawei_servicexx.com

----------xx_core_service

----------xx_other

----------go.mod

You need to set *GOPATH* to *${Directory where Go is located}* and add the following content to **go.mod**:

```
replace gitee.com/opengauss/openGauss-connector-go-pq => ../gitee.com/opengauss/openGauss-connector-go-pq
```

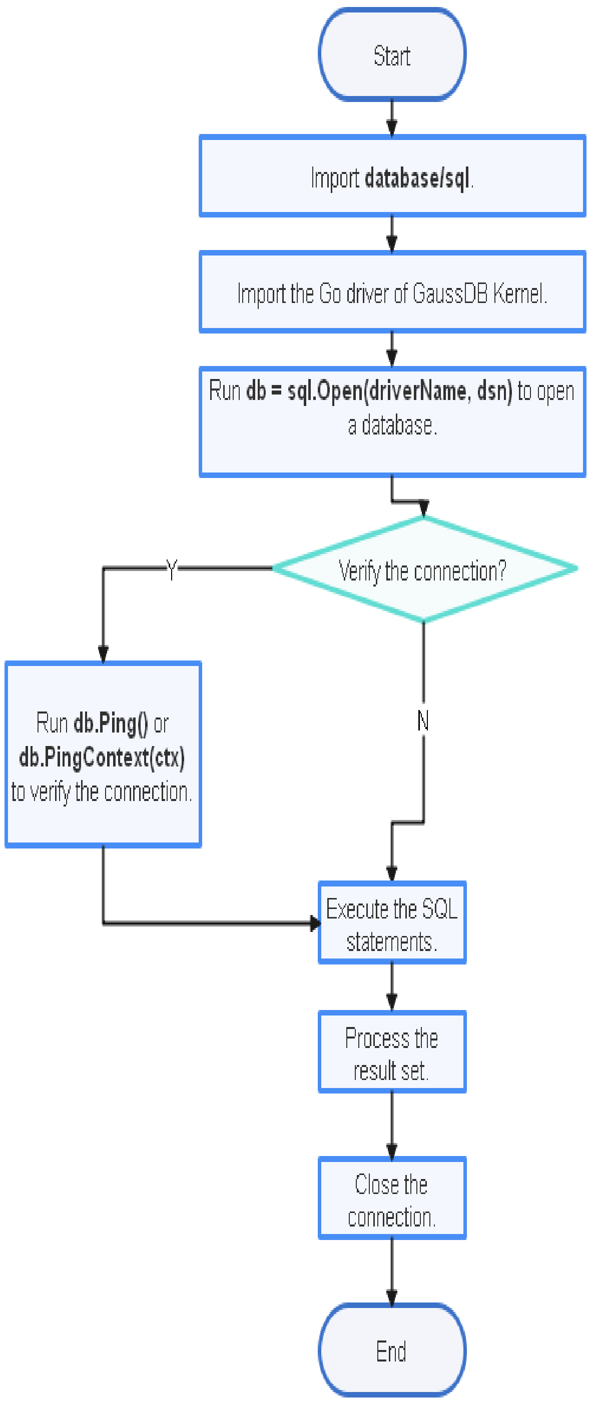
The preceding information indicates that all **gitee.com/opengauss/openGauss-connector-go-pq** in the code is imported locally and the dependency is not downloaded from the proxy.

If you do not want to reconstruct the go mod project, set **GO111MODULE** to **off** and manually download the dependency packages **xerrors** and **pbkdf2** to the directory specified by *${GOPATH}* which is at the same level as the driver root directory and service code.

5.7.3 Development Process

The Go driver of the database complies with the rule of the Go language third-party library. You only need to import the driver to the application program and save the driver code in the directory specified by *GOPATH*.

Figure 5-5 Application development process based on Go


Start
Import **database/sql**.
Import the Go driver of GaussDB Kernel.
Run **db = sql.Open(driverName, dsn)** to open a database.
Verify the connection?
Y
N
Run **db.Ping()** or **db.PingContext(ctx)** to verify the connection.
Execute the SQL statements.
Process the result set.
Close the connection.
End

5.7.4 Connecting to the Database

When you call the standard SQL API **open** of the Go language to create a database connection, a connected object is returned to transfer the driver name and description string.

Function Prototype

The Go driver provides the following method to generate a database connected object:

```
func Open(driverName, dataSourceName string) (*DB, error)
```

Parameter description:

- **driverName** indicates the driver name. The database driver name is **opengauss**, which is compatible with Postgres.
- **dataSourceName** indicates the data source to be connected. The value can be in DSN or URL format.
  - DSN format: key1 = value1 key2 = value2.... Different groups of keywords are separated by space. The space on the left and right of the equal sign (=) is optional.

    Example:

    Single IP address: host = 127.0.0.1 port = 1611 user = GaussDB password = GaussDB_Kernel dbname = postgres sslmode = disable connect_timeout = 100

    Multiple IP addresses: host = 127.0.0.1,127.0.0.2 port = 1611,1622 user = GaussDB password = GaussDB_Kernel dbname = postgres sslmode = disable connect_timeout = 100
  - URL format: driverName://[userspec@][hostspec][/dbname][?paramspec]

    In the preceding information, **driverName** indicates the driver name. The database driver name is **opengauss**, which is compatible with Postgres and PostgreSQL.

    userspec indicates user[:password]. When a URL is used for connection, the password cannot contain separators in the URL string. If the password contains separators, the DSN format is recommended.

    hostspec indicates [host][:port][, ...].

    dbname indicates the database name. Note: The initial user cannot be used for remote login. **paramspec** indicates name=value[&...].

    Example:

    Single IP address: opengauss://user:password@127.0.0.1:1611/postgres?sslmode=disable&connect_timeout=100

    Multiple IP addresses: opengauss://user:password@127.0.0.1:1611,127.0.0.2:1622/postgres?sslmode=disable&connect_timeout=100

NOTICE

- In the DSN format, if there are multiple IP addresses:
  - When the value of **num(ip)** is the same as that of **num(port)**, the IP address matches the port number.
  - When the value of **num(ip)** is greater than that of **num(port)**, the IP address that cannot match the port number matches the first port number. For example, the matching result of host = 127.0.0.1, 127.0.0.2,127.0.0.3 port = 1611,1622 is 127.0.0.1:1611,127.0.0.2:1622,127.0.0.3:1611.
  - If the value of **num(ip)** is smaller than that of **num(port)**, the extra port numbers are discarded. For example, the matching result of host = 127.0.0.1, 127.0.0.2,127.0.0.3 port = 1611,1622,1633,1644 is 127.0.0.1:1611,127.0.0.2:1622,127.0.0.3:1633.
- In the URL format, if there are multiple IP addresses:
  - In the URL, ***ip*:*port*** must appear in pairs, that is, the value of **num(ip)** is the same as that of **num(port)**. Use commas (,) to separate multiple pairs. Example: opengauss://user:password@127.0.0.1:1611,127.0.0.2:1622,127.0.0.3:1611/postgres
  - The URL contains only multiple IP addresses. The port number is specified by the environment variable or uses the default value **5432**. For example, with opengauss://user:password@127.0.0.1,127.0.0.2,127.0.0.3/postgres and the environment variable *PGPORT* set to **1611,1622**, the matching result is 127.0.0.1:1611,127.0.0.2:1622,127.0.0.3:1611. If the environment variable is not set, the matching result is 127.0.0.1:5432,127.0.0.2:5432,127.0.0.3:5432.

Parameters

Table 5-18 Database connection parameters

Parameter	Description
host	IP address of the host server, which can also be specified by the environment variable *${PGHOST}*
port	Port number of the host server, which can also be specified by the environment variable *${PGPORT}*
dbname	Database name, which can also be specified by the environment variable *${PGDATABASE}*
user	Username to be connected, which can also be specified by the environment variable *${PGUSER}*
password	Password of the user to be connected
connect_timeout	Timeout interval for connecting to the server, which can also be specified by the environment variable *${PGCONNECT_TIMEOUT}*

sslmode	SSL encryption mode, which can also be specified by the environment variable *${PGSSLMODE}* Value range: ● **disable**: SSL connection is disabled. ● **allow**: If the database server requires SSL connection, SSL connection can be enabled. However, authenticity of the database server will not be verified. ● **prefer**: If the database supports SSL connection, SSL connection is preferred. However, authenticity of the database server will not be verified. ● **require**: SSL connection is required and data is encrypted. However, authenticity of the database server will not be verified. ● **verify-ca**: SSL connection is required, and whether the server certificate is issued by a trusted CA is verified. ● **verify-full**: SSL connection is required, and whether the server certificate is issued by a trusted CA and whether the host name of the server is the same as that in the certificate are verified.
sslkey	Key location of the client certificate. If SSL connection is required and this parameter is not specified, you can set the environment variable *${PGSSLKEY}* to specify the location.
sslcert	File name of the client SSL certificate, which can also be specified by the environment variable *${PGSSLCERT}*
sslrootcert	Name of the file that contains the SSL CA certificate, which can also be specified by the environment variable *${PGSSLROOTCERT}*
sslcrl	File name of the SSL CRL. If a certificate listed in this file exists, the server certificate authentication will be rejected and the connection will fail. The value can also be specified by the environment variable *${PGSSLCRL}*.
sslpassword	Passphrase used to decrypt a key into plaintext. If this parameter is specified, the SSL key is an encrypted file. Currently, the SSL key supports DES encryption and AES encryption. **NOTE** The DES encryption algorithm has lower security and poses security risks. Therefore, you are advised to use a more secure encryption algorithm.

disable_prepared_binary_result	The value of this parameter is a string. If it is set to **yes**, the connection should not use the binary format when the query results are received from prepared statements. This parameter is used only for debugging. Value range: **yes** and **no**.
binary_parameters	Specifies whether **[]byte** is always sent in binary format. The value is a string. Value range: **yes** and **no**. If this parameter is set to **yes**, you are advised to bind parameters based on **[] byte** to reduce internal type conversion.
target_session_attrs	Connection type of the database, which can also be specified by the environment variable *${PGTARGETSESSIONATTRS}*. This parameter is used to identify the primary and standby nodes. There are six value options, namely, **any**, **master**, **slave**, **preferSlave**, **read-write**, and **read-only**. The default value is **any**. ● **any**: attempts to connect to any DN in the URL connection string. ● **master**: attempts to connect to a primary DN in the URL connection string. If the primary DN cannot be found, an exception is thrown. ● **slave**: attempts to connect to a standby DN in the URL connection string. If the standby DN cannot be found, an exception is thrown. ● **preferSlave**: attempts to connect to a standby DN (if available) in the URL connection string. Otherwise, it connects to the primary DN. ● **read-write**: specifies that only the primary DN can be connected. ● **read-only**: specifies that only the standby DN can be connected.
loggerLevel	Log level, which is used to print debugging information. The value can also be specified by the environment variable *${PGLOGGERLEVEL}*. The value can be **trace**, **debug**, **info**, **warn**, **error**, or **none**, in descending order of priority.
application_name	Name of the Go driver that is being connected. The default value is **go-driver**. You are not advised to configure this parameter.

RuntimeParams	Runtime parameter to be set to the session default value on a connection, for example, **search_path**, **application_name**, or **timezone**. For details about the parameters, see the default settings of the client connection. You can run the **SHOW** command to check whether the parameters are set successfully.

Example 1:

```
// The following code uses an ip:port as an example.
func main() {
 str := "host=127.0.0.1 port=1611 user=testuser password=Gauss_234 dbname=postgres
sslmode=disable"   // DSN connection string
 // str := "opengauss://testuser:Gauss_234@127.0.0.1:1611/postgres?sslmode=disable"   // URL connection
string
 db, err:= sql.Open("opengauss", str)
 if err != nil {
  log.Fatal(err)
    }
    defer db.Close()

    err = db.Ping()
    if err != nil {
        log.Fatal(err)
    }

    sqls := []string {
        "drop table if exists testExec",
        "create table testExec(f1 int, f2 varchar(20), f3 number, f4 timestamptz, f5 boolean)",
        "insert into testExec values(1, 'abcdefg', 123.3, '2022-02-08 10:30:43.31 +08', true)",
        "insert into testExec values(:f1, :f2, :f3, :f4, :f5)",
    }

    inF1 := []int{2, 3, 4, 5, 6}
    intF2 := []string{"hello world", "Huawei", "Beijing 2022 Winter Olympics", "nanjing", "Research Center"}
    intF3 := []float64{641.43, 431.54, 5423.52, 665537.63, 6503.1}
    intF4 := []time.Time{
        time.Date(2022, 2, 8, 10, 35, 43, 623431, time.Local),
        time.Date(2022, 2, 10, 19, 11, 54, 353431, time.Local),
        time.Date(2022, 2, 12, 6, 11, 15, 636431, time.Local),
        time.Date(2022, 2, 14, 4, 51, 22, 747653, time.Local),
        time.Date(2022, 2, 16, 13, 45, 55, 674636, time.Local),
    }
    intF5 := []bool{false, true, false, true, true}

    for _, s := range sqls {
        if strings.Contains(s, ":f") {
            for i, _ := range inF1 {
                _, err := db.Exec(s, inF1[i], intF2[i], intF3[i], intF4[i], intF5[i])
                if err != nil {
                    log.Fatal(err)
                }
            }
        } else {
            _, err = db.Exec(s)
            if err != nil {
                log.Fatal(err)
            }
        }
    }

    var f1 int
    var f2 string
    var f3 float64
    var f4 time.Time
    var f5 bool
    err = db.QueryRow("select * from testExec").Scan(&f1, &f2, &f3, &f4, &f5)
    if err != nil {
        log.Fatal(err)
    } else {
        fmt.Printf("f1:%v, f2:%v, f3:%v, f4:%v, f5:%v\n", f1, f2, f3, f4, f5)
    }

    row, err :=db.Query("select * from testExec where f1 > :1", 1)
    if err != nil {
        log.Fatal(err)
    }
    defer row.Close()

    for row.Next() {
        err = row.Scan(&f1, &f2, &f3, &f4, &f5)
        if err != nil {
            log.Fatal(err)
        } else {
            fmt.Printf("f1:%v, f2:%v, f3:%v, f4:%v, f5:%v\n", f1, f2, f3, f4, f5)
        }
    }
}
```

Example 2:

```
// The following code uses multiple ip:port as an example.
func main() {
    ctx := context.Background()
    ctx2SecondTimeout, cancelFunc2SecondTimeout := context.WithTimeout(ctx, 2 * time.Second)
    defer cancelFunc2SecondTimeout()

    str := "host=127.0.0.1,127.0.0.2,127.0.0.3 port=1611,1622 user=testuser password=Gauss_234
dbname=postgres sslmode = disable"     // DSN connection string
    // str := "opengauss://testuser:Gauss_234@127.0.0.1:1611,127.0.0.2:1612,127.0.0.3/postgres?
sslmode=disable"    // URL connection string
    db, err:= sql.Open("opengauss", str)
    if err != nil {
        log.Fatal(err)
    }
    defer db.Close()

    // Ping database connection with 2 second timeout
    err = db.PingContext(ctx2SecondTimeout)
    if err != nil {
        log.Fatal(err)
    }

    sqls := []string {
        "drop table if exists testExecContext",
        "create table testExecContext(f1 int, f2 varchar(20), f3 number, f4 timestamptz, f5 boolean)",
        "insert into testExecContext values(1, 'abcdefg', 123.3, '2022-02-08 10:30:43.31 +08', true)",
        "insert into testExecContext values(:f1, :f2, :f3, :f4, :f5)",
    }

    inF1 := []int{2, 3, 4, 5, 6}
    intF2 := []string{"hello world", "Huawei", "Beijing 2022 Winter Olympics", "nanjing", "Research Center"}
    intF3 := []float64{641.43, 431.54, 5423.52, 665537.63, 6503.1}
    intF4 := []time.Time{
        time.Date(2022, 2, 8, 10, 35, 43, 623431, time.Local),
        time.Date(2022, 2, 10, 19, 11, 54, 353431, time.Local),
        time.Date(2022, 2, 12, 6, 11, 15, 636431, time.Local),
        time.Date(2022, 2, 14, 4, 51, 22, 747653, time.Local),
        time.Date(2022, 2, 16, 13, 45, 55, 674636, time.Local),
    }
    intF5 := []bool{false, true, false, true, true}
    for _, s := range sqls {
        if strings.Contains(s, ":f") {
            for i, _ := range inF1 {
                _, err := db.ExecContext(ctx2SecondTimeout, s, inF1[i], intF2[i], intF3[i], intF4[i], intF5[i])
                if err != nil {
                    log.Fatal(err)
                }
            }
        } else {
            _, err = db.ExecContext(ctx2SecondTimeout, s)
            if err != nil {
                log.Fatal(err)
            }
        }
    }

    var f1 int
    var f2 string
    var f3 float64
    var f4 time.Time
    var f5 bool
    err = db.QueryRowContext(ctx2SecondTimeout, "select * from testExecContext").Scan(&f1, &f2, &f3, &f4,
&f5)
    if err != nil {
        log.Fatal(err)
    } else {
        fmt.Printf("f1:%v, f2:%v, f3:%v, f4:%v, f5:%v\n", f1, f2, f3, f4, f5)
    }

    row, err :=db.QueryContext(ctx2SecondTimeout, "select * from testExecContext where f1 > :1", 1)
    if err != nil {
        log.Fatal(err)
    }
    defer row.Close()

    for row.Next() {
        err = row.Scan(&f1, &f2, &f3, &f4, &f5)
        if err != nil {
            log.Fatal(err)
        } else {
            fmt.Printf("f1:%v, f2:%v, f3:%v, f4:%v, f5:%v\n", f1, f2, f3, f4, f5)
        }
    }
}
```

5.7.5 Connecting to the Database (Using SSL)

The Go driver supports SSL connections to the database. After the SSL mode is enabled, if the Go driver connects to the database server in SSL mode, the Go driver uses the standard TLS 1.3 protocol by default, and the TLS version must be 1.2 or later. This section describes how applications establish an SSL connection to the database by using the Go driver. Before the connection, you must obtain the certificates and private keys required for the client and server. For details about obtaining these files, see related documents and commands of OpenSSL.

NOTE

In SSL-based certificate authentication mode, you do not need to specify the user password in the connection string.

Configuring the Client

Upload the certificate files **client.key**, **client.crt**, and **cacert. pem** to the client.

Example 1:

```
// Take two-way authentication as an example.
func main() {
 dsnStr := "host=127.0.0.1 port=1611 user=testuser password=Gauss_234 dbname=postgres sslcert=certs/
client.crt sslkey=certs/client.key sslpassword=abcd "
 parameters := []string {
  " sslmode=require",
  " sslmode=verify-ca sslrootcert=certs/cacert.pem",
 }

 for _, param := range parameters {
  db, err:= sql.Open("opengauss", dsnStr+param)
  if err != nil {
   log.Fatal(err)
  }

  var f1 int
  err = db.QueryRow("select 1").Scan(&f1)
  if err != nil {
   log.Fatal(err)
  } else {
   fmt.Printf("RESULT: select 1: %d\n", f1)
  }

  db.Close()
 }
}
```

Example 2:

```
// Verify sslpassword.
func main() {
 dsnStr := "host=127.0.0.1 port=1611 user=testuser password=Gauss_234 dbname=postgres"

 connStrs := []string {
  " sslmode=verify-ca sslcert=certs/client_rsa.crt sslkey=certs/client_rsa.key sslpassword=abcd
sslrootcert=certs/cacert_rsa.pem",
  " sslmode=verify-ca sslcert=certs/client_ecdsa.crt sslkey=certs/client_ecdsa.key sslpassword=abcd
sslrootcert=certs/cacert_ecdsa.pem",
 }
 for _, connStr := range connStrs {
  db, err := sql.Open("opengauss", dsnStr + connStr)
  if err != nil {
   log.Fatal(err)
        }
        var f1 int
        err = db.QueryRow("select 1").Scan(&f1)
        if err != nil {
            if !strings.HasPrefix(err.Error(), "connect failed.") {
                log.Fatal(err)
            }
        }
        db.Close()
    }
}
```

5.7.6 Go APIs

For details, see **Go**.

6 Database Security Management

6.1 Checking the Number of Database Connections

Background

If the number of connections reaches its upper limit, new connections cannot be created. Therefore, if a user fails to connect a database, the administrator must check whether the number of connections has reached the upper limit. The following are details about database connections:

- The maximum number of global connections is specified by the **max_connections** parameter.
- The number of a user's connections is specified by **CONNECTION LIMIT connlimit** in the **CREATE ROLE** statement and can be changed using **CONNECTION LIMIT connlimit** in the **ALTER ROLE** statement.
- The number of a database's connections is specified by the **CONNECTION LIMIT connlimit** parameter in the **CREATE DATABASE** statement.

Procedure

Step 1 Connect to a database. For details, see **Connecting to a Database**.

Step 2 View the upper limit of the number of global connections.

```
openGauss=# SHOW max_connections;
 max_connections
-----------------
 800
(1 row)
```

800 is the maximum number of session connections.

Step 3 View the number of connections that have been used.

For details, see **Table 6-1**.

NOTICE

Except for database and usernames that are enclosed in double quotation marks (") during creation, uppercase letters are not allowed in the database and usernames in the commands in the following table.

Table 6-1 Viewing the number of session connections

Description	Command

View the maximum number of sessions connected to a specific user.	Run the following commands to view the upper limit of the number of **omm**'s session connections. **–1** indicates that no upper limit is set for the number of **omm**'s session connections.
	`openGauss=# SELECT ROLNAME,ROLCONNLIMIT FROM PG_ROLES WHERE ROLNAME='omm';`
	` rolname  | rolconnlimit`
	`----------+--------------`
	` omm |           -1`
	`(1 row)`

View the number of session connections that have been used by a user.	Run the following commands to view the number of session connections that have been used by **omm**. **1** indicates the number of session connections that have been used by **omm**.
	`openGauss=# CREATE OR REPLACE VIEW DV_SESSIONS AS`
	`          SELECT`
	`           sa.sessionid AS SID,`
	`           0::integer AS SERIAL#,`
	`           sa.usesysid AS USER#,`
	`           ad.rolname AS USERNAME`
	`           FROM pg_stat_get_activity(NULL) AS sa`
	`           LEFT JOIN pg_authid ad ON(sa.usesysid = ad.oid)`
	`           WHERE sa.application_name <> 'JobSchedul`
	`openGauss=# SELECT COUNT(*) FROM DV_SESSIONS WHERE USERNAME='omm';`
	
	` count`
	`-------`
	`     1`
	`(1 row)`

View the maximum number of sessions connected to a specific database.	Run the following commands to view the upper limit of the number of **postgres**'s session connections. **–1** indicates that no upper limit is set for the number of **postgres**'s session connections.
	`openGauss=# SELECT DATNAME,DATCONNLIMIT FROM PG_DATABASE WHERE DATNAME='postgres';`
	
	` datname  | datconnlimit`
	`----------+--------------`
	` postgres |           -1`
	`(1 row)`

Description	Command

View the number of session connections that have been used by a specific database.	Run the following commands to view the number of session connections that have been used by **postgres**. **1** indicates the number of session connections that have been used by **postgres**.
	`openGauss=# SELECT COUNT(*) FROM PG_STAT_ACTIVITY WHERE DATNAME='postgres';`
	` count`
	`-------`
	`     1`
	`(1 row)`

View the number of session connections that have been used by all users.	Run the following commands to view the number of session connections that have been used by all users:
	`openGauss=# CREATE OR REPLACE VIEW DV_SESSIONS AS`
	`          SELECT`
	`           sa.sessionid AS SID,`
	`           0::integer AS SERIAL#,`
	`           sa.usesysid AS USER#,`
	`           ad.rolname AS USERNAME`
	`           FROM pg_stat_get_activity(NULL) AS sa`
	`           LEFT JOIN pg_authid ad ON(sa.usesysid = ad.oid)`
	`           WHERE sa.application_name <> 'JobSchedul`
	`openGauss=# SELECT COUNT(*) FROM DV_SESSIONS;`
	` count`
	`-------`
	`    10`
	`(1 row)`

----End

6.2 Managing Users and Their Permissions

6.2.1 Default Permission Mechanism

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

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

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

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

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

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

You are not advised to modify the permissions on system catalogs or system views.

6.2.2 Administrator

Initial User

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

An initial user bypasses all permission checks. You are advised to use an initial user as a database administrator only for database management other than service running.

System Administrator

A system administrator is an account with the **SYSADMIN** attribute. By default, a system administrator has the same permissions as the object owner but does not have the object permissions in the **dbe_perf** schema or the permission to use Roach to perform backup and restoration.

To create a database administrator, connect to the database as an administrator and run the **CREATE USER** or **ALTER USER** statement with **SYSADMIN** specified.

```
openGauss=# CREATE USER sysadmin WITH SYSADMIN password "xxxxxxxxx";
```

or

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

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

Monitor Administrator

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

To create a monitoring administrator, connect to the database as a system administrator and run the **CREATE USER** or **ALTER USER** statement with **MONADMIN** specified.

```
openGauss=# CREATE USER monadmin WITH MONADMIN password "xxxxxxxxx";
```

Alternatively,

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

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

O&M Administrator

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

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

```
openGauss=# CREATE USER opradmin WITH OPRADMIN password "xxxxxxxxx";
```

Alternatively,

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

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

Security Policy Administrator

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

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

```
openGauss=# CREATE USER poladmin WITH POLADMIN password "xxxxxxxxx";
```

Alternatively,

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

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

6.2.3 Separation of Duties

Descriptions in **Default Permission Mechanism** and **Administrator** are about the initial situation after the database system is created. By default, a system administrator with the **SYSADMIN** attribute has the highest-level permissions.

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

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

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

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

WARNING

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

After separation of duties, the system administrator does not have permissions for non-system schemas of other users. Therefore, the system administrator cannot access the objects in other users' schemas before being granted the permissions. For details about permission changes before and after enabling separation of duties, see **Table 6-2** and **Table 6-3**.

Table 6-2 Default user permissions

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

Table 6-3 Changes in permissions after separation of duties

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

NOTICE

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

6.2.4 Users

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

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

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

Adding, Modifying, and Deleting Users

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

  For example, create a user **joe** and set the **CREATEDB** attribute for the user.
  ```
  openGauss=# CREATE USER joe WITH CREATEDB PASSWORD "xxxxxxxxx";
  CREATE ROLE
  ```
- To create a system administrator, use the **CREATE USER** statement with the **SYSADMIN** parameter.
- To delete an existing user, use **DROP USER**.
- To change a user account (for example, rename the user or change the password), use **ALTER USER**.
- To view the user list, query the view **PG_USER**:
  ```
  openGauss=# SELECT * FROM pg_user;
  ```
- To view user attributes, query the system catalog **PG_AUTHID**.
  ```
  openGauss=# SELECT * FROM pg_authid;
  ```

Private Users

If multiple service departments use different database user accounts to perform service operations and a database maintenance department at the same level uses database administrator accounts to perform maintenance operations, service departments may require that database administrators, without specific authorization, can perform the DROP, ALTER, and TRUNCATE operations on their data but cannot perform the INSERT, DELETE, UPDATE, SELECT, and COPY operations on the data. That is, the management permissions of database administrators for tables need to be isolated from their access permissions to improve the data security of common users.

In **Separation of Duties** mode, a database administrator does not have permissions for the tables in schemas of other users. In this case, database administrators have neither management permissions nor access permissions, which does not meet the requirements of the service departments mentioned above. Therefore, GaussDB provides private users to solve the problem. That is, create private users with the **INDEPENDENT** attribute in non-separation-of-duties mode. Users with the **CREATEROLE** permission or the system administrator permission can create private users or change the attributes of common users to private users. Common users can also change their own attributes to private users.

```
openGauss=# CREATE USER user_independent WITH INDEPENDENT IDENTIFIED BY "1234@abc";
```

System administrators can manage (DROP, ALTER, and TRUNCATE) table objects of private users but cannot access (INSERT, DELETE, SELECT, UPDATE, COPY, GRANT, REVOKE, and ALTER OWNER) the objects before being authorized.

NOTICE

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

Permanent User

GaussDB provides the permanent user solution. That is, create a permanent user with the **PERSISTENCE** attribute.

```
openGauss=# CREATE USER user_persistence WITH PERSISTENCE IDENTIFIED BY "1234@abc";
```

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

6.2.5 Roles

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

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

To view all roles, query the system catalog **PG_ROLES**.

```
SELECT * FROM PG_ROLES;
```

Adding, Modifying, and Deleting Roles

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

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

Built-in Roles

GaussDB provides a group of default roles whose names start with **gs_role_**. These roles are provided to access to specific, typically high-privileged operations. You can grant these roles to other users or roles within the database so that they can use specific functions. These roles should be given with great care to ensure that they are used where they are needed. **Table 6-4** describes the permissions of built-in roles.

Table 6-4 Permission description of built-in roles

Roles	Permission
gs_role_copy_files	Permission to run the **copy... to/from filename** command. However, the GUC parameter **enable_copy_server_files** must be set first to enable the function of copying server files.
gs_role_signal_backend	Permission to call the **pg_cancel_backend**, **pg_terminate_backend**, and **pg_terminate_session** functions to cancel or terminate other sessions. However, this role cannot perform operations on sessions of the initial user or **PERSISTENCE** user.
gs_role_tablespace	Permission to create a tablespace.
gs_role_replication	Permission to call logical replication functions, such as **kill_snapshot**, **pg_create_logical_replication_slot**, **pg_create_physical_replication_slot**, **pg_drop_replication_slot**, **pg_replication_slot_advance**, **pg_create_physical_replication_slot_extern**, **pg_logical_slot_get_changes**, **pg_logical_slot_peek_changes**, **pg_logical_slot_get_binary_changes**, and **pg_logical_slot_peek_binary_changes**.
gs_role_account_lock	Permission to lock and unlock users. However, this role cannot lock or unlock the initial user or users with the **PERSISTENCE** attribute.
gs_role_pldebugger	Permission to debug functions in **dbe_pldebugger**.
gs_role_directory_create	Permission to create directory objects. However, this role needs to enable the GUC parameter **enable_access_server_directory** first.
gs_role_directory_drop	Permission to delete directory objects. However, this role needs to enable the GUC parameter **enable_access_server_directory** first.

The restrictions on built-in roles are as follows:

- The role names starting with **gs_role_** are reserved for built-in roles in the database. Do not create users or roles starting with **gs_role_** or rename existing users or roles to names starting with **gs_role_**.
- Do not perform the **ALTER** or **DROP** operation on built-in roles.
- By default, built-in roles do not have the **LOGIN** permission and do not have preset passwords.

- The **gsql** meta-commands **\du** and **\dg** do not display information about built-in roles. However, if *pattern* is specified as a specific built-in role, the information is displayed.
- When separation-of-duty is disabled, the initial user, users with the **SYSADMIN** permission, and users with the **ADMIN OPTION** built-in role permission have the permission to perform **GRANT** and **REVOKE** operations on built-in roles. When separation of duty is enabled, the initial user and users with the **ADMIN OPTION** built-in role permission have the permission to perform **GRANT** and **REVOKE** operations on built-in roles. Example:
```
GRANT gs_role_signal_backend TO user1;
REVOKE gs_role_signal_backend FROM user1;
```

6.2.6 Schemas

Schemas function as models. Schema management allows multiple users to use the same database without mutual impacts, to organize database objects as manageable logical groups, and to add third-party applications to the same schema without causing conflicts.

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

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

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

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

Creating, Modifying, and Deleting Schemas

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

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

Search Path

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

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

  Run the following command to set **search_path** to **myschema** and **public** (**myschema** will be searched first):
```
openGauss=# SET SEARCH_PATH TO myschema, public;
SET
```

6.2.7 Setting User Permissions

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

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

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

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

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

     ```
     openGauss=# CREATE ROLE lily WITH CREATEDB PASSWORD "xxxxxxxxx";
     ```
  b. Grant object permissions to the role by using **GRANT**.

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

     ```
     openGauss=# GRANT USAGE ON SCHEMA tpcds TO lily;
     openGauss=# GRANT SELECT ON TABLE tpcds.web_returns to lily;
     ```
  c. Grant the role permissions to a user.

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

     NOTE

     When the permissions of a role are granted to a user, the attributes of the role are not transferred together.
- To revoke user permissions, use **REVOKE**.

6.2.8 Row-Level Security Policy

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

NOTICE

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

6.2.9 Setting Security Policies

6.2.9.1 Setting Account Security Policies

Background

For data security purposes, GaussDB provides a series of security measures, such as automatically locking and unlocking accounts, manually locking and unlocking abnormal accounts, and deleting accounts that are no longer used.

Automatically Locking and Unlocking Accounts

- If the number of incorrect password attempts (**failed_login_attempts**) of an account reaches the upper limit (**10** by default), the system automatically locks the account. Smaller parameter values result in higher account security. However, if the value of this parameter is set too small, inconvenience may occur.
- If the time during which a user is locked exceeds the preset value (**password_lock_time**, one day by default), the system automatically unlocks the user. Larger parameter values bring higher account security. However, if the value of this parameter is set too large, inconvenience may occur.

  NOTE

  - The integer part of the **password_lock_time** parameter indicates the number of days, and the decimal part can be converted into hours, minutes, and seconds. For example, **password_lock_time=1.5** indicates one day and 12 hours.
  - If the **failed_login_attempts** parameter is set to **0**, an account is never locked due to incorrect password attempts. If the **password_lock_time** parameter is set to **0**, an account is quickly unlocked after it is locked due to incorrect password attempts. Therefore, only when both parameters are set to positive values, the following operations can be performed: password failure check, account locking, and account unlocking.
  - The default values of the two parameters meet the security requirements. You can change the parameter values as needed for higher security. You are advised to retain the default values.

Manually Locking and Unlocking Accounts

Once detecting that an account is stolen or the account is used to access the database without being authorized, administrators can manually lock the account. Administrators can manually unlock the account if the account becomes normal again.

For details about how to create a user, see **Users**. To manually lock and unlock user **joe**, run commands in the following format:

- To manually lock the account, run the following command:
```
openGauss=# ALTER USER joe ACCOUNT LOCK;
ALTER ROLE
```
- To manually unlock the account, run the following command:
```
openGauss=# ALTER USER joe ACCOUNT UNLOCK;
ALTER ROLE
```

Deleting Accounts That Are No Longer Used

Administrators can delete an account that is no longer used. This operation cannot be rolled back.

When an account to be deleted is in the active state, it is deleted after the session is disconnected.

For example, if you want to delete account **joe**, run the following command:

```
openGauss=# DROP USER joe  CASCADE;
DROP ROLE
```

6.2.9.2 Setting the Validity Period of an Account

Precautions

When creating a user, you need to specify the validity period of the user, including the start time and end time.

To enable a user not within the validity period to use its account, set a new validity period.

Procedure

Step 1 Connect to a database. For details, see **Connecting to a Database**.

Step 2 Run the following command to create a user and specify the start time and end time:

```
openGauss=# CREATE USER joe WITH PASSWORD 'xxxxxxxxx' VALID BEGIN '2015-10-10 08:00:00' VALID
UNTIL '2016-10-10 08:00:00';
```

The user is created if the following information is displayed:

```
CREATE ROLE
```

Step 3 If the user is not within the specified validity period, run the following command to set the start time and end time of a new validity period:

```
openGauss=# ALTER USER joe WITH VALID BEGIN '2016-11-10 08:00:00' VALID UNTIL '2017-11-10
08:00:00';
```

The start time and end time of the new validity period is set successfully if the following information is displayed:

```
ALTER ROLE
```

----End

NOTE

- If **VALID BEGIN** and **VALID UNTIL** are not specified in the **CREATE ROLE** syntax, the start time and end time of a user's operation are not limited.
- If **VALID BEGIN** and **VALID UNTIL** are not specified in the **ALTER ROLE** syntax, the start time and end time of a user's operation are not modified and the original settings are used.

6.2.9.3 Setting Password Security Policies

Procedure

User passwords are stored in the system catalog **pg_authid**. To prevent password leakage, GaussDB encrypts user passwords before storing them. The encryption algorithm is determined by the configuration parameter **password_encryption_type**.

- If parameter **password_encryption_type** is set to **0**, passwords are encrypted using MD5. The MD5 encryption algorithm is not recommended because it has lower security and poses security risks.
- If parameter **password_encryption_type** is set to **1**, passwords are encrypted using SHA-256 and MD5. The MD5 encryption algorithm is not recommended because it has lower security and poses security risks.
- If parameter **password_encryption_type** is set to **2**, passwords are encrypted using SHA-256. This is the default configuration.
- If parameter **password_encryption_type** is set to **3**, passwords are encrypted using SM3.

Step 1 Connect to a database. For details, see **Connecting to a Database**.

Step 2 View the configured encryption algorithm.

```
openGauss=# SHOW password_encryption_type;
 password_encryption_type
--------------------------
 2
(1 row)
```

If **0** or **1** is displayed, run the **\q** command to exit the database.

Step 3 Change the value of **password_encryption_type** to ensure that passwords are encrypted using a secure encryption algorithm.

NOTICE

To prevent password leakage, when running **CREATE USER/ROLE** to create a database user, do not specify the **UNENCRYPTED** attribute. In this way, the password of the newly created user must be encrypted for storage.

Step 4 Configure password security parameters.

- Password complexity

  You need to specify a password when initializing a database, creating a user, or modifying a user. The password must meet the complexity check rules (see **password_policy**). Otherwise, you are prompted to enter the password again.

  – If parameter **password_policy** is set to **1**, the default password complexity rule is used to check passwords.

  – If parameter **password_policy** is set to **0**, the password complexity is not verified. You are not advised to set the parameter to this value because it is insecure. Password complexity is skipped only if the **password_policy** parameter is set to **0** for all database nodes.

  Configure the **password_policy** parameter.

The password complexity and length requirements are as follows:

- Minimum number of uppercase letters (A-Z) (**password_min_uppercase**)
- Minimum number of lowercase letters (a-z) (**password_min_lowercase**)
- Minimum number of digits (0-9) (**password_min_digital**)
- Minimum number of special characters (**password_min_special**) (**Table 6-5** lists special characters.)
- Minimum length of a password (**password_min_length**)
- Maximum length of a password (**password_max_length**)

  NOTE

  For details, see "GUC Parameters" in *Developer Guide*.

- A password must contain at least three types of the characters: uppercase letters, lowercase letters, digits, and special characters.
- A password is case insensitive and cannot be the username or the username spelled backwards.
- A new password cannot be the current password and the current password spelled backwards.
- A password cannot be a weak password.
  - Weak passwords are weak passwords that are easy to crack. The definition of weak passwords may vary with users or user groups. Users can define their own weak passwords.
  - Passwords in the weak password dictionary are stored in the **gs_global_config** system catalog. When a user is created or modified, the password set by the user is compared with the password stored in the weak password dictionary. If the password is matched, a message is displayed, indicating that the password is weak and the password fails to be set.
  - The weak password dictionary is empty by default. You can add or delete weak passwords using the following syntax:

    CREATE WEAK PASSWORD DICTIONARY WITH VALUES ('password1'), ('password2');

    DROP WEAK PASSWORD DICTIONARY;

- Password reuse

  An old password can be reused only when it meets the requirements on reuse days (**password_reuse_time**) and reuse times (**password_reuse_max**). **Table 6-6** lists the parameter configurations.

  NOTE

  The default values of the **password_reuse_time** and **password_reuse_max** parameters are **60** and **0**, respectively. Large values of the two parameters bring higher security. However, if the values of the parameters are set too large, inconvenience may occur. The default values of the two parameters meet the security requirements. You can change the parameter values as needed for higher security.

  Configure the **password_reuse_time** parameter.

  Configure the **password_reuse_max** parameter.

- Password validity period

  A validity period (**password_effect_time**) is set for each database user password. If the password is about to expire (**password_notify_time**), the system displays a message to remind the user to change it upon login.

  NOTE

  Considering the usage and service continuity of a database, the database still allows a user to log in after the password expires. A password change notification is displayed every time the user logs in to the database until the password is changed.

  Configure the **password_effect_time** parameter.

  Configure the **password_notify_time** parameter.

- Password change
  - During database installation, an OS user with the same name as the initial user is created. The password of the OS user needs to be periodically changed for account security.

    To change the password of user **user1**, run the following command:

    ```
    passwd user1
    ```

    Change the password as prompted.
  - System administrators and common users need to periodically change their passwords to prevent the accounts from being stolen.

    For example, to change the password of user **user1**, connect to the database as the system administrator and run the following commands:

    ```
    openGauss=# ALTER USER user1 IDENTIFIED BY "1234@abc" REPLACE "5678@def";
    ALTER ROLE
    ```

    NOTE

    1234@abc and **5678@def** represent the new password and the original password of user **user1**, respectively. If the new password does not have the required complexity, the change will not take effect.
  - Administrators can change their own and common users' passwords. If common users forget their passwords, they can ask administrators to change the passwords.

    To change the password of user **joe**, run the following command:

    ```
    openGauss=# ALTER USER joe IDENTIFIED BY "abc@1234";
    ALTER ROLE
    ```

  NOTE

  - System administrators are not allowed to change passwords for each other.
  - A system administrator can change the password of a common user without being required to provide the common user's old password.
  - A system administrator can change their own password but is required to provide the old password.

- Password verification

  Password verification is required when you set the user or role in the current session. If the entered password is inconsistent with the stored password of the user, an error is reported.

  If user **joe** needs to be set, run the following command:

  ```
  openGauss=# SET ROLE joe PASSWORD "abc@1234";
  ERROR:  Invalid username/password,set role denied.
  ```

Table 6-5 Special characters

No.	Character	No.	Character	No.	Character	No.	Character
1	~	9	*	17	|	25	<
2	!	10	(	18	[	26	.
3	@	11	)	19	{	27	>
4	#	12	-	20	}	28	/
5	$	13	_	21	]	29	?
6	%	14	=	22	;	-	-
7	^	15	+	23	:	-	-
8	&	16	\	24	,	-	-

Table 6-6 Parameter description for reuse days and reuse times

Parameter	Value Range	Description
Number of days during which a password cannot be reused (**password_reuse_time**)	Positive number or **0**. The integral part of a positive number indicates the number of days and its decimal part can be converted into hours, minutes, and seconds. The default value is **60**.	• If the parameter value is changed to a smaller one, new passwords will be checked based on the new parameter value. • If the parameter value is changed to a larger one (for example, changed from **a** to **b**), the historical passwords before **b** days probably can be reused because these historical passwords may have been deleted. New passwords will be checked based on the new parameter value. **NOTE** The absolute time is used. Historical passwords are recorded using absolute time and unaffected by local time changes.

Parameter	Value Range	Description
Number of consecutive times that a password cannot be reused (**password_reuse_max**)	Positive integer or **0**. The value **0** indicates that the number of consecutive times that a password cannot be reused is not checked.	● If the parameter value is changed to a smaller one, new passwords will be checked based on the new parameter value. ● If the parameter value is changed to a larger one (for example, changed from **a** to **b**), the historical passwords before the last **b** passwords probably can be reused because these historical passwords may have been deleted. New passwords will be checked based on the new parameter value.

Step 5 Set user password expiration.

When creating a user, a user with the CREATEROLE permission can force the user password to expire. After logging in to the database for the first time, a new user can perform query operations only after changing the password. The command format is as follows:

```
openGauss=# CREATE USER joe PASSWORD "abc@1234" EXPIRED;
CREATE ROLE
```

A user with the CREATEROLE permission can force a user password to expire or force a user to change the forcibly expired password. The command format is as follows:

```
openGauss=# ALTER USER joe PASSWORD EXPIRED;
ALTER ROLE
openGauss=# ALTER USER joe PASSWORD "abc@2345" EXPIRED;
ALTER ROLE
```

NOTE

- After a user whose password is invalid logs in to the database, the system prompts the user to change the password when the user performs a simple or extended query. The user can then execute the statement after changing the password.
- Only initial users, system administrators (with the sysadmin permission), and users who have the permission to create users (with the CREATEROLE permission) can invalidate user passwords. System administrators can invalidate their own passwords or the passwords of other system administrators. The initial user password cannot be invalidated.

----End

6.3 Configuring Database Audit

6.3.1 Overview

Background

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

You need to know the following about the audit function:

- The overall audit switch **audit_enabled** supports dynamic loading. After you change the switch status when the database is running, the change takes effect immediately and you do not need to restart the database. Its default value is **on**, indicating that the audit function is enabled.
- In addition to the overall audit switch, each audit item has an independent switch. An audit item is available only after its own switch is turned on.
- The switch of each audit supports dynamic loading. After changing the audit switch status when the database is running, the modification takes effect immediately without restarting the database.

Table 6-7 describes the audit items supported by GaussDB.

Table 6-7 Audit items

Configuration Item	Description
User login and logout audit	Parameter: **audit_login_logout** Its default value is **7**, which indicates that the function of user login and logout audit is enabled. **0** indicates that the function of user login and logout audit is disabled. Other values are not recommended.
Database startup, stop, recovery, and switchover audit	Parameter: **audit_database_process** Its default value is **1**, which indicates that the audit of database startup, stop, recovery, and switchover is enabled.
User locking and unlocking audit	Parameter: **audit_user_locked** Its default value is **1**, which indicates that the audit of user locking and unlocking is enabled.
Unauthorized access audit	Parameter: **audit_user_violation** Its default value is **0**, which indicates that the audit of unauthorized access disabled.
Permission granting and revoking audit	Parameter: **audit_grant_revoke** Its default value is **1**, which indicates that the audit of permission granting and revoking is enabled.
Audit of CREATE, ALTER, and DROP operations on a database object	Parameter: **audit_system_object** Its default value is **67121159**, which indicates that the CREATE, ALTER, and DROP operations only on databases, schemas, users, data sources, and SQL patches are audited.

Configuration Item	Description
Audit of INSERT, UPDATE, and DELETE operations on a specific table	Parameter: **audit_dml_state** Its default value is **0**, which indicates that the audit of DML operations (except SELECT) on a specific table is disabled.
SELECT audit	Parameter: **audit_dml_state_select** Its default value is **0**, which indicates that the audit of SELECT operations is disabled.
COPY audit	Parameter: **audit_copy_exec** Its default value is **1**, which indicates that the audit of COPY operations is enabled.
Execution of stored procedures and customized functions	Parameter: **audit_function_exec** Its default value is **0**, which indicates that no execution audit logs of stored procedures and customized functions are recorded.
SET audit	Parameter: **audit_set_parameter** Its default value is **0**, which indicates that the audit of SET operations is disabled.
Transaction ID record	Parameter: **audit_xid_info** Its default value is **0**, which indicates that the function of recording transaction IDs in audit logs is disabled.

For details about security-related parameters, see **Table 6-8**.

Table 6-8 Security-related parameters

Parameter	Description
ssl	Whether the SSL connection is enabled.
require_ssl	Whether the server requires the SSL connection.
ssl_ciphers	Encryption algorithm list supported by the SSL
ssl_cert_file	File containing the SSL server certificate
ssl_key_file	File containing the SSL private key
ssl_ca_file	File containing CA information
ssl_crl_file	File containing CRL information
password_policy	Whether to check the password complexity.
password_reuse_time	Whether to check the reuse days of a new password.

Parameter	Description
password_reuse_max	Whether to check the reuse times of a new password.
password_lock_time	Duration before a locked account is automatically unlocked
failed_login_attempts	If the number of consecutive login attempts with incorrect passwords reaches this value, the account is locked.
password_encryption_type	Password storage encryption mode
password_min_uppercase	Minimum number of uppercase letters in a password
password_min_lowercase	Minimum number of lowercase letters in a password
password_min_digital	Minimum number of digits in a password
password_min_special	Minimum number of special characters in a password
password_min_length	Minimum password length **NOTE** The value of this parameter must be less than or equal to that of **password_max_length**. Otherwise, a password length error message is displayed upon all password-related operations.
password_max_length	Maximum password length **NOTE** The value of this parameter must be greater than or equal to that of **password_min_length**. Otherwise, a password length error message is displayed upon all password-related operations.
password_effect_time	Password validity period
password_notify_time	Number of days prior to account password expiration that a user is notified
audit_enabled	Whether the audit process is enabled or disabled
audit_directory	Audit file storage directory
audit_data_format	Audit log file format. Currently, only the binary format is supported
audit_rotation_interval	Time interval of creating an audit log file. If the difference between the current time and the time when the previous audit log file is created is greater than the value of **audit_rotation_interval**, a new audit log file will be generated.

Parameter	Description
audit_rotation_size	Maximum capacity of an audit log file. If the total number of messages in an audit log exceeds the value of **audit_rotation_size**, the server will generate a new audit log file.
audit_resource_policy	Policy for determining whether audit logs are preferentially stored by space or time. **on** indicates that audit logs are preferentially stored by space.
audit_file_remain_time	Minimum duration required for recording audit logs. This parameter is valid only when **audit_resource_policy** is set to **off**.
audit_space_limit	Maximum total size of audit log files in a disk
audit_file_remain_threshold	Maximum number of audit files in the audit directory
audit_login_logout	Whether to audit user logins (including login successes and failures) and logouts
audit_database_process	Whether to audit database startup, stop, switchover, and restoration operations
audit_user_locked	Whether to audit database user locking and unlocking
audit_user_violation	Whether to audit beyond-authority operations of a database user
audit_grant_revoke	Whether to audit user permission granting and reclaiming operations
audit_system_object	Whether to audit the CREATE, ALTER, and DROP operations on database objects
audit_dml_state	Whether to audit the INSERT, UPDATE, and DELETE operations on a specific table
audit_dml_state_select	Whether to audit the SELECT operation
audit_copy_exec	Whether to audit the COPY operation
audit_function_exec	Whether to record audit information during execution of stored procedures, anonymous blocks, or customized functions (excluding system functions)
audit_set_parameter	Whether to audit the SET operation
enableSeparationOfDuty	Whether the separation of duty is enabled
session_timeout	If the duration of a connection session exceeds the parameter value, the session is automatically disconnected.

Parameter	Description
auth_iteration_count	Number of iterations during the generation of encrypted information for authentication

Procedure

Step 1 Connect to a database. For details, see **Connecting to a Database**.

Step 2 Check the status of the overall audit switch.

1. Run the **show** command to view the value of **audit_enabled**.
```
openGauss=# SHOW audit_enabled;
```
If the status is **off**, run the **\q** command to exit the database.
2. Set **audit_enabled** to **on** to enable the audit function. The parameter setting takes effect immediately.

Step 3 Configure specific audit items based on **Table 6-7**.

NOTE

- After the audit function is enabled, user operations can be recorded into an audit file.
- The default parameter value of each audit item meets security standards. You can enable audit functions as needed, but system performance may be affected.

----End

6.3.2 Querying Audit Results

Prerequisites

- Audit has been enabled.
- Audit of required items has been enabled.
- The database is running properly and a series of addition, modification, deletion, and query operations have been executed in the database. Otherwise, no audit result is generated.
- Audit logs are separately recorded on the database nodes.

Background

- Only users with the **AUDITADMIN** permission can view audit records. For details about database users and how to create users, see **Users**.
- The SQL function **pg_query_audit** is provided by the database for audit query. Its syntax is as follows:
```
pg_query_audit(timestamptz startime,timestamptz endtime,audit_log)
```
startime and **endtime** indicate the start time and end time of the audit record, respectively. **audit_log** indicates the physical file path of the queried audit logs. If **audit_log** is not specified, the audit log information of the current instance is queried.

NOTE

The difference between the values of the **startime** and **endtime** parameters indicates the query period, which can be any value ranging from 00:00:00 of the **startime** parameter to 23:59:59 of the **endtime** parameter. Therefore, the **startime** and **endtime** parameters must be properly set to ensure that the required audit information is displayed.

Procedure

Step 1 Connect to a database. For details, see **Connecting to a Database**.

Step 2 Run the following command to query the audit record:

```
openGauss=# select * from pg_query_audit('2021-03-04 08:00:00','2021-03-04 17:00:00');
```

The command output is similar to the following:

```
          time          |      type      | result | userid | username  | database  |     client_conninfo     |
object_name    |   detail_info    |     node_name      |            thread_id            | local_port | remote_port
------------------------+----------------+--------+--------+-----------+-----------+-------------------------
+-------------------+--------------+-------------------+--------------------------------+------------+-------------
2021-03-04 08:00:08+08 | login_success  | ok     | 10     | omm  | postgres | gsql@::1 | postgres    | login
db(postgres) success, SSL=off | dn_6001_6002_6003 | 140477687527168@668131208211425 |17778      |
46946
```

This record indicates that user **omm** logs in to database **postgres** at the time specified by the **time** column. After the host specified by **log_hostname** is started and a client is connected to its IP address, the host name found by reverse DNS resolution is displayed following the at sign (@) in the value of **client_conninfo**.

NOTE

SSL information is recorded at the end of the audit log **detail_info**. **SSL=on** indicates that the client is connected using SSL, and **SSL=off** indicates that the client is not connected using SSL.

----End

6.3.3 Maintaining Audit Logs

Prerequisites

You have the audit permission.

Background

- **Table 6-9** lists the configuration parameters related to audit logs and the parameter descriptions.

Table 6-9 Configuration parameters of audit logs

Parameter	Description	Default Value
audit_directory	Audit file storage directory	*$GAUSSLOG*/**pg_audit**

Parameter	Description	Default Value
audit_resource_policy	Policy for saving audit logs	**on** (indicating that the space configuration policy is used)
audit_space_limit	Maximum storage space occupied by audit files	1 GB
audit_file_remain_time	Minimum period for storing audit log files	90
audit_file_remain_threshold	Maximum number of audit files in the audit directory	1048576

- The **pg_delete_audit** function is used to delete audit logs and is an internal SQL function of the database. Its syntax is as follows:

```
pg_delete_audit(timestamp startime,timestamp endtime)
```

  startime and **endtime** indicate the audit record start time and end time, respectively.
- Audit content is commonly recorded to database tables or OS files. **Table 6-10** lists the advantages and disadvantages of the two record methods.

Table 6-10 Comparison between the two record methods

Mode	Benefit	Drawback
Record in tables	Users do not need to maintain audit logs.	Any users having certain permissions to access database objects can access the audit tables. If a user illegally performs operations on the audit tables, the audit records may become inaccurate.
Record in OS files	This method has higher security because a user with the permission to access the database may not have the permission to access the OS files.	Users need to maintain audit logs.

For database security purposes, GaussDB adopts the second method to save audit results for reliability.

Procedure

Step 1 Connect to a database. For details, see **Connecting to a Database**.

Step 2 Select a method to maintain audit logs.

- Automatic deletion of audit logs

  If the storage space occupied by audit files or the number of audit files reaches the upper limit, the system automatically deletes the oldest audit files and records deletion information to audit logs.

  NOTE

  By default, the allowed maximum storage space that can be occupied by audit files is set to **1024 MB**. Users can set the value as required.

  Configure the allowed maximum size of storage space occupied by audit files (**audit_space_limit**).

  Configure the maximum number of audit files (**audit_file_remain_threshold**).

- Manually back up audit files.

  If the storage space occupied by audit files or the number of audit logs exceeds the threshold specified by the configuration file, the system automatically deletes the oldest audit files. Therefore, you are advised to periodically save important audit logs.

  a. Run the **show** command to view the directory (**audit_directory**) where audit files are saved.
     ```
     openGauss=# SHOW audit_directory;
     ```
  b. Copy the entire audit directory elsewhere to save it.

- Manually delete audit logs.

  Run the **pg_delete_audit** command to manually delete the audit records generated during a specified period of time.

  Example: Manually delete the audit records generated from September 20, 2012 to September 21, 2012.
  ```
  openGauss=# SELECT pg_delete_audit('2012-09-20 00:00:00','2012-09-21 23:59:59');
  ```

----End

7 API Reference

7.1 JDBC

This section describes common JDBC interfaces. For more interfaces, check JDK1.8 (software package) and JDBC 4.0.

7.1.1 java.sql.Connection

This section describes **java.sql.Connection**, the API for connecting to a database.

Table 7-1 Support status for java.sql.Connection

Method Name	Return Type	JDBC 4 Is Supported Or Not
abort(Executor executor)	void	Yes
clearWarnings()	void	Yes
close()	void	Yes
commit()	void	Yes
createArrayOf(String typeName, Object[] elements)	Array	Yes
createBlob()	Blob	Yes
createClob()	Clob	Yes
createSQLXML()	SQLXML	Yes
createStatement()	Statement	Yes
createStatement(int resultSetType, int resultSetConcurrency)	Statement	Yes

Method Name	Return Type	JDBC 4 Is Supported Or Not
createStatement(int resultSetType, int resultSetConcurrency, int resultSetHoldability)	Statement	Yes
getAutoCommit()	Boolean	Yes
getCatalog()	String	Yes
getClientInfo()	Properties	Yes
getClientInfo(String name)	String	Yes
getHoldability()	int	Yes
getMetaData()	DatabaseMetaData	Yes
getNetworkTimeout()	int	Yes
getSchema()	String	Yes
getTransactionIsolation()	int	Yes
getTypeMap()	Map<String,Class<?>>	Yes
getWarnings()	SQLWarning	Yes
isClosed()	Boolean	Yes
isReadOnly()	Boolean	Yes
isValid(int timeout)	Boolean	Yes
nativeSQL(String sql)	String	Yes
prepareCall(String sql)	CallableStatement	Yes
prepareCall(String sql, int resultSetType, int resultSetConcurrency)	CallableStatement	Yes
prepareCall(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability)	CallableStatement	Yes
prepareStatement(String sql)	PreparedStatement	Yes
prepareStatement(String sql, int autoGeneratedKeys)	PreparedStatement	Yes
prepareStatement(String sql, int[] columnIndexes)	PreparedStatement	Yes

Method Name	Return Type	JDBC 4 Is Supported Or Not
prepareStatement(String sql, int resultSetType, int resultSetConcurrency)	PreparedStatement	Yes
prepareStatement(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability)	PreparedStatement	Yes
prepareStatement(String sql, String[] columnNames)	PreparedStatement	Yes
releaseSavepoint(Savepoint savepoint)	void	Yes
rollback()	void	Yes
rollback(Savepoint savepoint)	void	Yes
setAutoCommit(boolean autoCommit)	void	Yes
setClientInfo(Properties properties)	void	Yes
setClientInfo(String name,String value)	void	Yes
setHoldability(int holdability)	void	Yes
setNetworkTimeout (Executor executor, int milliseconds)	void	Yes
setReadOnly(boolean readOnly)	void	Yes
setSavepoint()	Savepoint	Yes
setSavepoint(String name)	Savepoint	Yes
setSchema(String schema)	void	Yes
setTransactionIsolation(int level)	void	Yes
setTypeMap(Map<String, Class<?>> map)	void	Yes

NOTICE

The AutoCommit mode is used by default within the API. If you disable it by running **setAutoCommit(false)**, all the statements executed later will be packaged in explicit transactions, and you cannot execute statements that cannot be executed within transactions.

7.1.2 java.sql.CallableStatement

This section describes **java.sql.CallableStatement**, the API for executing the stored procedure.

Table 7-2 Support status for java.sql.CallableStatement

Method Name	Return Type	JDBC 4 Is Supported Or Not
getArray(int parameterIndex)	Array	Yes
getBigDecimal(int parameterIndex)	BigDecimal	Yes
getBlob(int parameterIndex)	Blob	Yes
getBoolean(int parameterIndex)	Boolean	Yes
getByte(int parameterIndex)	byte	Yes
getBytes(int parameterIndex)	byte[]	Yes
getClob(int parameterIndex)	Clob	Yes
getDate(int parameterIndex)	Date	Yes
getDate(int parameterIndex, Calendar cal)	Date	Yes
getDouble(int parameterIndex)	double	Yes
getFloat(int parameterIndex)	float	Yes
getInt(int parameterIndex)	int	Yes
getLong(int parameterIndex)	long	Yes
getObject(int parameterIndex)	Object	Yes
getObject(int parameterIndex, Class<T> type)	Object	Yes
getShort(int parameterIndex)	short	Yes

Method Name	Return Type	JDBC 4 Is Supported Or Not
getSQLXML(int parameterIndex)	SQLXML	Yes
getString(int parameterIndex)	String	Yes
getNString(int parameterIndex)	String	Yes
getTime(int parameterIndex)	Time	Yes
getTime(int parameterIndex, Calendar cal)	Time	Yes
getTimestamp(int parameterIndex)	Timestamp	Yes
getTimestamp(int parameterIndex, Calendar cal)	Timestamp	Yes
registerOutParameter(int parameterIndex, int type)	void	Yes
registerOutParameter(int parameterIndex, int sqlType, int type)	void	Yes
wasNull()	Boolean	Yes

NOTE

- The batch operation of statements containing OUT parameter is not allowed.
- The following methods are inherited from **java.sql.Statement**: **close**, **execute**, **executeQuery**, **executeUpdate**, **getConnection**, **getResultSet**, **getUpdateCount**, **isClosed**, **setMaxRows**, and **setFetchSize**.
- The following methods are inherited from **java.sql.PreparedStatement**: **addBatch**, **clearParameters**, **execute**, **executeQuery**, **executeUpdate**, **getMetaData**, **setBigDecimal**, **setBoolean**, **setByte**, **setBytes**, **setDate**, **setDouble**, **setFloat**, **setInt**, **setLong**, **setNull**, **setObject**, **setString**, **setTime**, and **setTimestamp**.
- The **registerOutParameter(int parameterIndex, int sqlType, int type)** method is used only to register the composite data type.

7.1.3 java.sql.DatabaseMetaData

This section describes **java.sql.DatabaseMetaData**, the API for defining database objects.

Table 7-3 Support status for java.sql.DatabaseMetaData

Method Name	Return Type	JDBC 4 Is Supported Or Not
allProceduresAreCallable()	Boolean	Yes
allTablesAreSelectable()	Boolean	Yes
autoCommitFailureClosesAll-ResultSets()	Boolean	Yes
dataDefinitionCausesTran-sactionCommit()	Boolean	Yes
dataDefinitionIgnoredIn-Transactions()	Boolean	Yes
deletesAreDetected(int type)	Boolean	Yes
doesMaxRowSizeInclude-Blobs()	Boolean	Yes
generatedKeyAlwaysRe-turned()	Boolean	Yes
getBestRowIdentifier(String catalog, String schema, String table, int scope, boolean nullable)	ResultSet	Yes
getCatalogs()	ResultSet	Yes
getCatalogSeparator()	String	Yes
getCatalogTerm()	String	Yes
getClientInfoProperties()	ResultSet	Yes
getColumnPrivileges(String catalog, String schema, String table, String columnNamePattern)	ResultSet	Yes
getConnection()	Connection	Yes
getCrossReference(String parentCatalog, String parentSchema, String parentTable, String foreignCatalog, String foreignSchema, String foreignTable)	ResultSet	Yes
getDefaultTransactionIsola-tion()	int	Yes

Method Name	Return Type	JDBC 4 Is Supported Or Not
getExportedKeys(String catalog, String schema, String table)	ResultSet	Yes
getExtraNameCharacters()	String	Yes
getFunctionColumns(String catalog, String schemaPattern, String functionNamePattern, String columnNamePattern)	ResultSet	Yes
getFunctions(String catalog, String schemaPattern, String functionNamePattern)	ResultSet	Yes
getIdentifierQuoteString()	String	Yes
getImportedKeys(String catalog, String schema, String table)	ResultSet	Yes
getIndexInfo(String catalog, String schema, String table, boolean unique, boolean approximate)	ResultSet	Yes
getMaxBinaryLiteralLength()	int	Yes
getMaxCatalogName-Length()	int	Yes
getMaxCharLiteralLength()	int	Yes
getMaxColumnName-Length()	int	Yes
getMaxColumnsInGroupBy()	int	Yes
getMaxColumnsInIndex()	int	Yes
getMaxColumnsInOrderBy()	int	Yes
getMaxColumnsInSelect()	int	Yes
getMaxColumnsInTable()	int	Yes
getMaxConnections()	int	Yes
getMaxCursorNameLength()	int	Yes
getMaxIndexLength()	int	Yes
getMaxLogicalLobSize()	default long	Yes

Method Name	Return Type	JDBC 4 Is Supported Or Not
getMaxProcedureName-Length()	int	Yes
getMaxRowSize()	int	Yes
getMaxSchemaName-Length()	int	Yes
getMaxStatementLength()	int	Yes
getMaxStatements()	int	Yes
getMaxTableNameLength()	int	Yes
getMaxTablesInSelect()	int	Yes
getMaxUserNameLength()	int	Yes
getNumericFunctions()	String	Yes
getPrimaryKeys(String catalog, String schema, String table)	ResultSet	Yes
getPartitionTablePrimary-Keys(String catalog, String schema, String table)	ResultSet	Yes
getProcedureColumns(String catalog, String schemaPattern, String procedureNamePattern, String columnNamePattern)	ResultSet	Yes
getProcedures(String catalog, String schemaPattern, String procedureNamePattern)	ResultSet	Yes
getProcedureTerm()	String	Yes
getSchemas()	ResultSet	Yes
getSchemas(String catalog, String schemaPattern)	ResultSet	Yes
getSchemaTerm()	String	Yes
getSearchStringEscape()	String	Yes
getSQLKeywords()	String	Yes
getSQLStateType()	int	Yes
getStringFunctions()	String	Yes

Method Name	Return Type	JDBC 4 Is Supported Or Not
getSystemFunctions()	String	Yes
getTablePrivileges(String catalog, String schemaPattern, String tableNamePattern)	ResultSet	Yes
getTimeDateFunctions()	String	Yes
getTypeInfo()	ResultSet	Yes
getUDTs(String catalog, String schemaPattern, String typeNamePattern, int[] types)	ResultSet	Yes
getURL()	String	Yes
getVersionColumns(String catalog, String schema, String table)	ResultSet	Yes
insertsAreDetected(int type)	Boolean	Yes
locatorsUpdateCopy()	Boolean	Yes
othersDeletesAreVisible(int type)	Boolean	Yes
othersInsertsAreVisible(int type)	Boolean	Yes
othersUpdatesAreVisible(int type)	Boolean	Yes
ownDeletesAreVisible(int type)	Boolean	Yes
ownInsertsAreVisible(int type)	Boolean	Yes
ownUpdatesAreVisible(int type)	Boolean	Yes
storesLowerCaseIdentifiers()	Boolean	Yes
storesMixedCaseIdentifiers()	Boolean	Yes
storesUpperCaseIdentifiers()	Boolean	Yes
supportsBatchUpdates()	Boolean	Yes
supportsCatalogsInDataMa-nipulation()	Boolean	Yes

Method Name	Return Type	JDBC 4 Is Supported Or Not
supportsCatalogsInIndexDefinitions()	Boolean	Yes
supportsCatalogsInPrivilegeDefinitions()	Boolean	Yes
supportsCatalogsInProcedureCalls()	Boolean	Yes
supportsCatalogsInTableDefinitions()	Boolean	Yes
supportsCorrelatedSubqueries()	Boolean	Yes
supportsDataDefinitionAndDataManipulationTransactions()	Boolean	Yes
supportsDataManipulationTransactionsOnly()	Boolean	Yes
supportsGetGeneratedKeys()	Boolean	Yes
supportsMixedCaseIdentifiers()	Boolean	Yes
supportsMultipleOpenResults()	Boolean	Yes
supportsNamedParameters()	Boolean	Yes
supportsOpenCursorsAcrossCommit()	Boolean	Yes
supportsOpenCursorsAcrossRollback()	Boolean	Yes
supportsOpenStatementsAcrossCommit()	Boolean	Yes
supportsOpenStatementsAcrossRollback()	Boolean	Yes
supportsPositionedDelete()	Boolean	Yes
supportsPositionedUpdate()	Boolean	Yes
supportsRefCursors()	Boolean	Yes
supportsResultSetConcurrency(int type, int concurrency)	Boolean	Yes
supportsResultSetType(int type)	Boolean	Yes

Method Name	Return Type	JDBC 4 Is Supported Or Not
supportsSchemasInIndexDefinitions()	Boolean	Yes
supportsSchemasInPrivilegeDefinitions()	Boolean	Yes
supportsSchemasInProcedureCalls()	Boolean	Yes
supportsSchemasInTableDefinitions()	Boolean	Yes
supportsSelectForUpdate()	Boolean	Yes
supportsStatementPooling()	Boolean	Yes
supportsStoredFunctionsUsingCallSyntax()	Boolean	Yes
supportsStoredProcedures()	Boolean	Yes
supportsSubqueriesInComparisons()	Boolean	Yes
supportsSubqueriesInExists()	Boolean	Yes
supportsSubqueriesInIns()	Boolean	Yes
supportsSubqueriesInQuantifieds()	Boolean	Yes
supportsTransactionIsolationLevel(int level)	Boolean	Yes
supportsTransactions()	Boolean	Yes
supportsUnion()	Boolean	Yes
supportsUnionAll()	Boolean	Yes
updatesAreDetected(int type)	Boolean	Yes
getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)	ResultSet	Yes
getColumns(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern)	ResultSet	Yes
getTableTypes()	ResultSet	Yes
getUserName()	String	Yes

Method Name	Return Type	JDBC 4 Is Supported Or Not
isReadOnly()	Boolean	Yes
nullsAreSortedHigh()	Boolean	Yes
nullsAreSortedLow()	Boolean	Yes
nullsAreSortedAtStart()	Boolean	Yes
nullsAreSortedAtEnd()	Boolean	Yes
getDatabaseProductName()	String	Yes
getDatabaseProductVersion()	String	Yes
getDriverName()	String	Yes
getDriverVersion()	String	Yes
getDriverMajorVersion()	int	Yes
getDriverMinorVersion()	int	Yes
usesLocalFiles()	Boolean	Yes
usesLocalFilePerTable()	Boolean	Yes
supportsMixedCaseIdentifiers()	Boolean	Yes
storesUpperCaseIdentifiers()	Boolean	Yes
storesLowerCaseIdentifiers()	Boolean	Yes
supportsMixedCaseQuotedIdentifiers()	Boolean	Yes
storesUpperCaseQuotedIdentifiers()	Boolean	Yes
storesLowerCaseQuotedIdentifiers()	Boolean	Yes
storesMixedCaseQuotedIdentifiers()	Boolean	Yes
supportsAlterTableWithAddColumn()	Boolean	Yes
supportsAlterTableWithDropColumn()	Boolean	Yes
supportsColumnAliasing()	Boolean	Yes
nullPlusNonNullIsNull()	Boolean	Yes
supportsConvert()	Boolean	Yes

Method Name	Return Type	JDBC 4 Is Supported Or Not
supportsConvert(int fromType, int toType)	Boolean	Yes
supportsTableCorrelationNames()	Boolean	Yes
supportsDifferentTableCorrelationNames()	Boolean	Yes
supportsExpressionsInOrderBy()	Boolean	Yes
supportsOrderByUnrelated()	Boolean	Yes
supportsGroupBy()	Boolean	Yes
supportsGroupByUnrelated()	Boolean	Yes
supportsGroupByBeyondSelect()	Boolean	Yes
supportsLikeEscapeClause()	Boolean	Yes
supportsMultipleResultSets()	Boolean	Yes
supportsMultipleTransactions()	Boolean	Yes
supportsNonNullableColumns()	Boolean	Yes
supportsMinimumSQLGrammar()	Boolean	Yes
supportsCoreSQLGrammar()	Boolean	Yes
supportsExtendedSQLGrammar()	Boolean	Yes
supportsANSI92EntryLevelSQL()	Boolean	Yes
supportsANSI92IntermediateSQL()	Boolean	Yes
supportsANSI92FullSQL()	Boolean	Yes
supportsIntegrityEnhancementFacility()	Boolean	Yes
supportsOuterJoins()	Boolean	Yes
supportsFullOuterJoins()	Boolean	Yes
supportsLimitedOuterJoins()	Boolean	Yes

Method Name	Return Type	JDBC 4 Is Supported Or Not
isCatalogAtStart()	Boolean	Yes
supportsSchemasInDataManipulation()	Boolean	Yes
supportsSavepoints()	Boolean	Yes
supportsResultSetHoldability(int holdability)	Boolean	Yes
getResultSetHoldability()	int	Yes
getDatabaseMajorVersion()	int	Yes
getDatabaseMinorVersion()	int	Yes
getJDBCMajorVersion()	int	Yes
getJDBCMinorVersion()	int	Yes

NOTE

If the value of **uppercaseAttributeName** is **true**, the following APIs convert the query result to uppercase letters. The conversion range is the same as that of the **toUpperCase** method in Java.

- public ResultSet getProcedures(String catalog, String schemaPattern, String procedureNamePattern)
- public ResultSet getProcedureColumns(String catalog, String schemaPattern, String procedureNamePattern, String columnNamePattern)
- public ResultSet getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)
- public ResultSet getSchemas(String catalog, String schemaPattern)
- public ResultSet getColumns(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern)
- public ResultSet getColumnPrivileges(String catalog, String schema, String table, String columnNamePattern)
- public ResultSet getTablePrivileges(String catalog, String schemaPattern, String tableNamePattern)
- public ResultSet getBestRowIdentifier(String catalog, String schema, String table, int scope, boolean nullable)
- public ResultSet getPrimaryKeys(String catalog, String schema, String table)
- protected ResultSet getImportedExportedKeys(String primaryCatalog, String primarySchema, String primaryTable, String foreignCatalog, String foreignSchema, String foreignTable)
- public ResultSet getIndexInfo(String catalog, String schema, String tableName, boolean unique, boolean approximate)
- public ResultSet getUDTs(String catalog, String schemaPattern, String typeNamePattern, int[] types)
- public ResultSet getFunctions(String catalog, String schemaPattern, String functionNamePattern)

CAUTION

The **getPartitionTablePrimaryKeys(String catalog, String schema, String table)** API is used to obtain the primary key column of a partitioned table that contains global indexes. The following is an example:

```
PgDatabaseMetaData dbmd = (PgDatabaseMetaData)conn.getMetaData();
dbmd.getPartitionTablePrimaryKeys("catalogName", "schemaName", "tableName");
```

7.1.4 java.sql.Driver

This section describes **java.sql.Driver**, the database driver API.

Table 7-4 Support status for java.sql.Driver

Method Name	Return Type	JDBC 4 Is Supported Or Not
acceptsURL(String url)	Boolean	Yes
connect(String url, Properties info)	Connection	Yes
jdbcCompliant()	Boolean	Yes
getMajorVersion()	int	Yes
getMinorVersion()	int	Yes
getParentLogger()	Logger	Yes
getPropertyInfo(String url, Properties info)	DriverPropertyInfo[]	Yes

7.1.5 java.sql.PreparedStatement

This section describes **java.sql.PreparedStatement**, the API for preparing statements.

Table 7-5 Support status for java.sql.PreparedStatement

Method Name	Return Type	JDBC 4 Is Supported Or Not
clearParameters()	void	Yes
execute()	Boolean	Yes
executeQuery()	ResultSet	Yes
excuteUpdate()	int	Yes
executeLargeUpdate()	long	No

Method Name	Return Type	JDBC 4 Is Supported Or Not
getMetaData()	ResultSetMetaData	Yes
getParameterMetaData()	ParameterMetaData	Yes
setArray(int parameterIndex, Array x)	void	Yes
setAsciiStream(int parameterIndex, InputStream x, int length)	void	Yes
setBinaryStream(int parameterIndex, InputStream x)	void	Yes
setBinaryStream(int parameterIndex, InputStream x, int length)	void	Yes
setBinaryStream(int parameterIndex, InputStream x, long length)	void	Yes
setBlob(int parameterIndex, InputStream inputStream)	void	Yes
setBlob(int parameterIndex, InputStream inputStream, long length)	void	Yes
setBlob(int parameterIndex, Blob x)	void	Yes
setCharacterStream(int parameterIndex, Reader reader)	void	Yes
setCharacterStream(int parameterIndex, Reader reader, int length)	void	Yes
setClob(int parameterIndex, Reader reader)	void	Yes

Method Name	Return Type	JDBC 4 Is Supported Or Not
setClob(int parameterIndex, Reader reader, long length)	void	Yes
setClob(int parameterIndex, Clob x)	void	Yes
setDate(int parameterIndex, Date x, Calendar cal)	void	Yes
setNull(int parameterIndex, int sqlType)	void	Yes
setNull(int parameterIndex, int sqlType, String typeName)	void	Yes
setObject(int parameterIndex, Object x)	void	Yes
setObject(int parameterIndex, Object x, int targetSqlType)	void	Yes
setObject(int parameterIndex, Object x, int targetSqlType, int scaleOrLength)	void	Yes
setSQLXML(int parameterIndex, SQLXML xmlObject)	void	Yes
setTime(int parameterIndex, Time x)	void	Yes
setTime(int parameterIndex, Time x, Calendar cal)	void	Yes
setTimestamp(int parameterIndex, Timestamp x)	void	Yes
setTimestamp(int parameterIndex, Timestamp x, Calendar cal)	void	Yes

Method Name	Return Type	JDBC 4 Is Supported Or Not
setUnicodeStream(int parameterIndex, InputStream x, int length)	void	Yes
setURL(int parameterIndex, URL x)	void	Yes
setBoolean(int parameterIndex, boolean x)	void	Yes
setBigDecimal(int parameterIndex, BigDecimal x)	void	Yes
setByte(int parameterIndex, byte x)	void	Yes
setBytes(int parameterIndex, byte[] x)	void	Yes
setDate(int parameterIndex, Date x)	void	Yes
setDouble(int parameterIndex, double x)	void	Yes
setFloat(int parameterIndex, float x)	void	Yes
setInt(int parameterIndex, int x)	void	Yes
setLong(int parameterIndex, long x)	void	Yes
setShort(int parameterIndex, short x)	void	Yes
setString(int parameterIndex, String x)	void	Yes
setNString(int parameterIndex, String x)	void	Yes
addBatch()	void	Yes
executeBatch()	int[]	Yes

NOTE

- Execute **addBatch()** and **execute()** only after running **clearBatch()**.
- Batch is not cleared by calling **executeBatch()**. Clear batch by explicitly calling **clearBatch()**.
- After bounded variables of a batch are added, if you want to reuse these values, you do not need to use **set*()** again. Instead, add a batch.
- The following methods are inherited from **java.sql.Statement**: **close**, **execute**, **executeQuery**, **executeUpdate**, **getConnection**, **getResultSet**, **getUpdateCount**, **isClosed**, **setMaxRows**, and **setFetchSize**.
- The **executeLargeUpdate()** method can only be used in JDBC 4.2 or later.

7.1.6 java.sql.ResultSet

This section describes **java.sql.ResultSet**, the API for execution result sets.

Table 7-6 Support status for java.sql.ResultSet

Method Name	Return Type	JDBC 4 Is Supported Or Not
absolute(int row)	Boolean	Yes
afterLast()	void	Yes
beforeFirst()	void	Yes
cancelRowUpdates()	void	Yes
clearWarnings()	void	Yes
close()	void	Yes
deleteRow()	void	Yes
findColumn(String columnLabel)	int	Yes
first()	Boolean	Yes
getArray(int columnIndex)	Array	Yes
getArray(String columnLabel)	Array	Yes
getAsciiStream(int columnIndex)	InputStream	Yes
getAsciiStream(String columnLabel)	InputStream	Yes
getBigDecimal(int columnIndex)	BigDecimal	Yes
getBigDecimal(String columnLabel)	BigDecimal	Yes

Method Name	Return Type	JDBC 4 Is Supported Or Not
getBinaryStream(int columnIndex)	InputStream	Yes
getBinaryStream(String columnLabel)	InputStream	Yes
getBlob(int columnIndex)	Blob	Yes
getBlob(String columnLabel)	Blob	Yes
getBoolean(int columnIndex)	Boolean	Yes
getBoolean(String columnLabel)	Boolean	Yes
getByte(int columnIndex)	byte	Yes
getBytes(int columnIndex)	byte[]	Yes
getByte(String columnLabel)	byte	Yes
getBytes(String columnLabel)	byte[]	Yes
getCharacterStream(int columnIndex)	Reader	Yes
getCharacterStream (String columnLabel)	Reader	Yes
getClob(int columnIndex)	Clob	Yes
getClob(String columnLabel)	Clob	Yes
getConcurrency()	int	Yes
getCursorName()	String	Yes
getDate(int columnIndex)	Date	Yes
getDate(int columnIndex, Calendar cal)	Date	Yes
getDate(String columnLabel)	Date	Yes
getDate(String columnLabel, Calendar cal)	Date	Yes

Method Name	Return Type	JDBC 4 Is Supported Or Not
getDouble(int columnIndex)	double	Yes
getDouble(String columnLabel)	double	Yes
getFetchDirection()	int	Yes
getFetchSize()	int	Yes
getFloat(int columnIndex)	float	Yes
getFloat(String columnLabel)	float	Yes
getInt(int columnIndex)	int	Yes
getInt(String columnLabel)	int	Yes
getLong(int columnIndex)	long	Yes
getLong(String columnLabel)	long	Yes
getMetaData()	ResultSetMetaData	Yes
getObject(int columnIndex)	Object	Yes
getObject(int columnIndex, Class<T> type)	<T> T	Yes
getObject(int columnIndex, Map<String, Class<?>> map)	Object	Yes
getObject(String columnLabel)	Object	Yes
getObject(String columnLabel, Class<T> type)	<T> T	Yes
getObject(String columnLabel, Map<String, Class<?>> map)	Object	Yes
getRow()	int	Yes
getShort(int columnIndex)	short	Yes

Method Name	Return Type	JDBC 4 Is Supported Or Not
getShort(String columnLabel)	short	Yes
getSQLXML(int columnIndex)	SQLXML	Yes
getSQLXML(String columnLabel)	SQLXML	Yes
getStatement()	Statement	Yes
getString(int columnIndex)	String	Yes
getString(String columnLabel)	String	Yes
getNString(int columnIndex)	String	Yes
getNString(String columnLabel)	String	Yes
getTime(int columnIndex)	Time	Yes
getTime(int columnIndex, Calendar cal)	Time	Yes
getTime(String columnLabel)	Time	Yes
getTime(String columnLabel, Calendar cal)	Time	Yes
getTimestamp(int columnIndex)	Timestamp	Yes
getTimestamp(int columnIndex, Calendar cal)	Timestamp	Yes
getTimestamp(String columnLabel)	Timestamp	Yes
getTimestamp(String columnLabel, Calendar cal)	Timestamp	Yes
getType()	int	Yes
getWarnings()	SQLWarning	Yes
insertRow()	void	Yes

Method Name	Return Type	JDBC 4 Is Supported Or Not
isAfterLast()	Boolean	Yes
isBeforeFirst()	Boolean	Yes
isClosed()	Boolean	Yes
isFirst()	Boolean	Yes
isLast()	Boolean	Yes
last()	Boolean	Yes
moveToCurrentRow()	void	Yes
moveToInsertRow()	void	Yes
next()	Boolean	Yes
previous()	Boolean	Yes
refreshRow()	void	Yes
relative(int rows)	Boolean	Yes
rowDeleted()	Boolean	Yes
rowInserted()	Boolean	Yes
rowUpdated()	Boolean	Yes
setFetchDirection(int direction)	void	Yes
setFetchSize(int rows)	void	Yes
updateArray(int columnIndex, Array x)	void	Yes
updateArray(String columnLabel, Array x)	void	Yes
updateAsciiStream(int columnIndex, InputStream x, int length)	void	Yes
updateAsciiStream(String columnLabel, InputStream x, int length)	void	Yes
updateBigDecimal(int columnIndex, BigDecimal x)	void	Yes
updateBigDecimal(String columnLabel, BigDecimal x)	void	Yes

Method Name	Return Type	JDBC 4 Is Supported Or Not
updateBinaryStream(int columnIndex, InputStream x, int length)	void	Yes
updateBinaryStream (String columnLabel, InputStream x, int length)	void	Yes
updateBoolean(int columnIndex, boolean x)	void	Yes
updateBoolean(String columnLabel, boolean x)	void	Yes
updateByte(int columnIndex, byte x)	void	Yes
updateByte(String columnLabel, byte x)	void	Yes
updateBytes(int columnIndex, byte[] x)	void	Yes
updateBytes(String columnLabel, byte[] x)	void	Yes
updateCharacterStream (int columnIndex, Reader x, int length)	void	Yes
updateCharacterStream (String columnLabel, Reader reader, int length)	void	Yes
updateDate(int columnIndex, Date x)	void	Yes
updateDate(String columnLabel, Date x)	void	Yes
updateDouble(int columnIndex, double x)	void	Yes
updateDouble(String columnLabel, double x)	void	Yes
updateFloat(int columnIndex, float x)	void	Yes
updateFloat(String columnLabel, float x)	void	Yes
updateInt(int columnIndex, int x)	void	Yes

Method Name	Return Type	JDBC 4 Is Supported Or Not
updateInt(String columnLabel, int x)	void	Yes
updateLong(int columnIndex, long x)	void	Yes
updateLong(String columnLabel, long x)	void	Yes
updateNull(int columnIndex)	void	Yes
updateNull(String columnLabel)	void	Yes
updateObject(int columnIndex, Object x)	void	Yes
updateObject(int columnIndex, Object x, int scaleOrLength)	void	Yes
updateObject(String columnLabel, Object x)	void	Yes
updateObject(String columnLabel, Object x, int scaleOrLength)	void	Yes
updateRow()	void	Yes
updateShort(int columnIndex, short x)	void	Yes
updateShort(String columnLabel, short x)	void	Yes
updateSQLXML(int columnIndex, SQLXML xmlObject)	void	Yes
updateSQLXML(String columnLabel, SQLXML xmlObject)	void	Yes
updateString(int columnIndex, String x)	void	Yes
updateString(String columnLabel, String x)	void	Yes
updateTime(int columnIndex, Time x)	void	Yes

Method Name	Return Type	JDBC 4 Is Supported Or Not
updateTime(String columnLabel, Time x)	void	Yes
updateTimestamp(int columnIndex, Timestamp x)	void	Yes
updateTimestamp(String columnLabel, Timestamp x)	void	Yes
wasNull()	Boolean	Yes

NOTE

- One statement cannot have multiple open ResultSets.
- The cursor that is used for traversing the ResultSet cannot be open after being committed.

7.1.7 java.sql.ResultSetMetaData

This section describes **java.sql.ResultSetMetaData**, which provides details about ResultSet object information.

Table 7-7 Support status for java.sql.ResultSetMetaData

Method Name	Return Type	JDBC 4 Is Supported Or Not
getCatalogName(int column)	String	Yes
getColumnClassName(int column)	String	Yes
getColumnCount()	int	Yes
getColumnDisplaySize(int column)	int	Yes
getColumnLabel(int column)	String	Yes
getColumnName(int column)	String	Yes
getColumnType(int column)	int	Yes
getColumnTypeName(int column)	String	Yes
getPrecision(int column)	int	Yes

Method Name	Return Type	JDBC 4 Is Supported Or Not
getScale(int column)	int	Yes
getSchemaName(int column)	String	Yes
getTableName(int column)	String	Yes
isAutoIncrement(int column)	Boolean	Yes
isCaseSensitive(int column)	Boolean	Yes
isCurrency(int column)	Boolean	Yes
isDefinitelyWritable(int column)	Boolean	Yes
isNullable(int column)	int	Yes
isReadOnly(int column)	Boolean	Yes
isSearchable(int column)	Boolean	Yes
isSigned(int column)	Boolean	Yes
isWritable(int column)	Boolean	Yes

NOTE

When **uppercaseAttributeName** is set to **true**, the following APIs convert the query result to uppercase letters. The conversion range is 26 English letters.

- public String getColumnName(int column)
- public String getColumnLabel(int column)

7.1.8 java.sql.Statement

This section describes **java.sql.Statement**, the interface for executing SQL statements.

Table 7-8 Support status for java.sql.Statement

Method Name	Return Type	JDBC 4 Is Supported Or Not
addBatch(String sql)	void	Yes
clearBatch()	void	Yes
clearWarnings()	void	Yes
close()	void	Yes
closeOnCompletion()	void	Yes

Method Name	Return Type	JDBC 4 Is Supported Or Not
execute(String sql)	Boolean	Yes
execute(String sql, int autoGeneratedKeys)	Boolean	Yes
execute(String sql, int[] columnIndexes)	Boolean	Yes
execute(String sql, String[] columnNames)	Boolean	Yes
executeBatch()	Boolean	Yes
executeQuery(String sql)	ResultSet	Yes
executeUpdate(String sql)	int	Yes
executeUpdate(String sql, int autoGeneratedKeys)	int	Yes
executeUpdate(String sql, int[] columnIndexes)	int	Yes
executeUpdate(String sql, String[] columnNames)	int	Yes
getConnection()	Connection	Yes
getFetchDirection()	int	Yes
getFetchSize()	int	Yes
getGeneratedKeys()	ResultSet	Yes
getMaxFieldSize()	int	Yes
getMaxRows()	int	Yes
getMoreResults()	boolean	Yes
getMoreResults(int current)	boolean	Yes
getResultSet()	ResultSet	Yes
getResultSetConcurren-cy()	int	Yes
getResultSetHoldabili-ty()	int	Yes

Method Name	Return Type	JDBC 4 Is Supported Or Not
getResultSetType()	int	Yes
getQueryTimeout()	int	Yes
getUpdateCount()	int	Yes
getWarnings()	SQLWarning	Yes
isClosed()	Boolean	Yes
isCloseOnCompletion()	Boolean	Yes
isPoolable()	Boolean	Yes
setCursorName(String name)	void	Yes
setEscapeProcessing (boolean enable)	void	Yes
setFetchDirection(int direction)	void	Yes
setMaxFieldSize(int max)	void	Yes
setMaxRows(int max)	void	Yes
setPoolable(boolean poolable)	void	Yes
setQueryTimeout(int seconds)	void	Yes
setFetchSize(int rows)	void	Yes
cancel()	void	Yes
executeLargeUpdate(String sql)	long	No
getLargeUpdateCount()	long	No
executeLargeBatch()	long	No
executeLargeUpdate(String sql, int autoGeneratedKeys)	long	No
executeLargeUpdate(String sql, int[] columnIndexes)	long	No
executeLargeUpdate(String sql, String[] columnNames)	long	No

NOTE

- Using setFetchSize can reduce the memory occupied by result sets on the client. Result sets are packaged into cursors and segmented for processing, which will increase the communication traffic between the database and the client, affecting performance.
- Database cursors are valid only within their transactions. If **setFetchSize** is set, set **setAutoCommit(false)** and commit transactions on the connection to flush service data to a database.
- **LargeUpdate** methods can only be used in JDBC 4.2 or later.

7.1.9 javax.sql.ConnectionPoolDataSource

This section describes **javax.sql.ConnectionPoolDataSource**, the API for data source connection pools.

Table 7-9 Support status for javax.sql.ConnectionPoolDataSource

Method Name	Return Type	JDBC 4 Is Supported Or Not
getPooledConnection()	PooledConnection	Yes
getPooledConnection(String user,String password)	PooledConnection	Yes

7.1.10 javax.sql.DataSource

This section describes **javax.sql.DataSource**, the interface for data sources.

Table 7-10 Support status for javax.sql.DataSource

Method Name	Return Type	Support JDBC 4
getConneciton()	Connection	Yes
getConnection(String username,String password)	Connection	Yes
getLoginTimeout()	int	Yes
getLogWriter()	PrintWriter	Yes
setLoginTimeout(int seconds)	void	Yes
setLogWriter(PrintWriter out)	void	Yes

7.1.11 javax.sql.PooledConnection

This section describes **javax.sql.PooledConnection**, the connection API created by a connection pool.

Table 7-11 Support status for javax.sql.PooledConnection

Method Name	Return Type	JDBC 4 Is Supported Or Not
addConnectionEventListener (ConnectionEventListener listener)	void	Yes
close()	void	Yes
getConnection()	Connection	Yes
removeConnectionEventListener (ConnectionEventListener listener)	void	Yes

7.1.12 javax.naming.Context

This section describes **javax.naming.Context**, the context interface for connection configuration.

Table 7-12 Support status for javax.naming.Context

Method Name	Return Type	Support JDBC 4
bind(Name name, Object obj)	void	Yes
bind(String name, Object obj)	void	Yes
lookup(Name name)	Object	Yes
lookup(String name)	Object	Yes
rebind(Name name, Object obj)	void	Yes
rebind(String name, Object obj)	void	Yes
rename(Name oldName, Name newName)	void	Yes
rename(String oldName, String newName)	void	Yes
unbind(Name name)	void	Yes
unbind(String name)	void	Yes

7.1.13 javax.naming.spi.InitialContextFactory

This section describes **javax.naming.spi.InitialContextFactory**, the initial context factory interface.

Table 7-13 Support status for javax.naming.spi.InitialContextFactory

Method Name	Return Type	Support JDBC 4
getInitialContext(Hashtable<?,?> environment)	Context	Yes

7.1.14 CopyManager

CopyManager is an API class provided by the JDBC driver in GaussDB. It is used to import data to GaussDB in batches.

Inheritance Relationship of CopyManager

The CopyManager class is in the **org.postgresql.copy** package and inherits the java.lang.Object class. The declaration of the class is as follows:

```
public class CopyManager
extends Object
```

Construction Method

public CopyManager(BaseConnection connection)

throws SQLException

Common Methods

Table 7-14 Common methods of CopyManager

Return Value	Method	Description	throws
CopyIn	copyIn(String sql)	-	SQLException
long	copyIn(String sql, InputStream from)	Uses **COPY FROM STDIN** to quickly load data to tables in the database from InputStream.	SQLException,IOException

Return Value	Method	Description	throws
long	copyIn(String sql, InputStream from, int bufferSize)	Uses **COPY FROM STDIN** to quickly load data to tables in the database from InputStream.	SQLException,IOException
long	copyIn(String sql, Reader from)	Uses **COPY FROM STDIN** to quickly load data to tables in the database from Reader.	SQLException,IOException
long	copyIn(String sql, Reader from, int bufferSize)	Uses **COPY FROM STDIN** to quickly load data to tables in the database from Reader.	SQLException,IOException
CopyOut	copyOut(String sql)	-	SQLException
long	copyOut(String sql, OutputStream to)	Sends the result set of **COPY TO STDOUT** from the database to the OutputStream class.	SQLException,IOException
long	copyOut(String sql, Writer to)	Sends the result set of **COPY TO STDOUT** from the database to the Writer class.	SQLException,IOException

7.1.15 PGReplicationConnection

PGReplicationConnection is an API class provided by the JDBC driver in GaussDB. It is used to implement functions related to logical replication.

Inheritance Relationship of PGReplicationConnection

PGReplicationConnection is a logical replication interface. Its implementation class is PGReplicationConnectionImpl, which is in the **org.postgresql.replication** package. The declaration of the class is as follows:

```
public class PGReplicationConnection implements PGReplicationConnection
```

Construction Method

public PGReplicationConnection(BaseConnection connection)

Common Methods

Table 7-15 Common methods of PGReplicationConnection

Return Value	Method	Description	Throws
ChainedCreateReplicationSlotBuilder	createReplication-Slot()	Creates a logical replication slot.	-
void	dropReplication-Slot(String slotName)	Deletes a logical replication slot.	SQLException,IOException
ChainedStreamBuilder	replicationStream()	Enables logical replication.	-

7.1.16 PGReplicationStream

PGReplicationStream is an API class provided by the GaussDB JDBC driver. It is used to operate logical replication streams.

Inheritance Relationship of PGReplicationStream

PGReplicationStream is a logical replication API. Its implementation class is V3PGReplicationStream, which is in the **org.postgresql.core.v3.replication** package. The declaration of the class is as follows:

```
public class V3PGReplicationStream implements PGReplicationStream
```

Constructor

public V3PGReplicationStream(CopyDual copyDual, LogSequenceNumber startLSN, long updateIntervalMs, ReplicationType replicationType)

Common Methods

Table 7-16 Common methods of PGReplicationConnection

Return Value	Method	Description	throws
void	close()	Ends the logical replication and releases resources.	SQLException

Return Value	Method	Description	throws
void	forceUpdateStatus()	Forcibly sends the LSN status received, flushed, and applied last time to the backend.	SQLException
LogSequenceNumber	getLastAppliedLSN()	Obtains the LSN when the primary node replays logs last time.	-
LogSequenceNumber	getLastFlushedLSN()	Obtains the LSN flushed by the primary node last time, that is, the LSN pushed by the current logic decoding.	-
LogSequenceNumber	getLastReceiveLSN()	Obtains the LSN received last time.	-
boolean	isClosed()	Determines whether the replication stream is disabled.	-
ByteBuffer	read()	Reads the next WAL record from the backend. If the data cannot be read, this method blocks the I/O read.	SQLException
ByteBuffer	readPending()	Reads the next WAL record from the backend. If the data cannot be read, this method does not block the I/O read.	SQLException
void	setAppliedLSN(LogSequenceNumber applied)	Sets the applied LSN.	-

Return Value	Method	Description	throws
void	setFlushedLSN(LogSequenceNumber flushed)	Sets the flushed LSN, which is sent to the backend at the next update to push the LSN on the server.	-

7.1.17 ChainedStreamBuilder

ChainedStreamBuilder is an API class provided by the GaussDB JDBC driver. It is used to build replication streams.

Inheritance Relationship of ChainedStreamBuilder

ChainedStreamBuilder is a logical replication API. Its implementation class is ReplicationStreamBuilder, which is in the **org.postgresql.replication.fluent** package. The declaration of the class is as follows:

```
public class ReplicationStreamBuilder implements ChainedStreamBuilder
```

Constructor

public ReplicationStreamBuilder(final BaseConnection connection)

Common Methods

Table 7-17 Common methods of ReplicationStreamBuilder

Return Value	Method	Description	throws
ChainedLogicalStreamBuilder	logical()	Creates a logical replication stream.	-
ChainedPhysical-StreamBuilder	physical()	Creates a physical replication stream.	-

7.1.18 ChainedCommonStreamBuilder

ChainedCommonStreamBuilder is an API class provided by the GaussDB JDBC driver. It is used to specify common parameters for logical and physical replication.

Inheritance Relationship of ChainedCommonStreamBuilder

ChainedCommonStreamBuilder is an API for logical replication. The implementation abstract class is AbstractCreateSlotBuilder. The inheritance class is

LogicalCreateSlotBuilder which is in the **org.postgresql.replication.fluent.logical** package. The declaration of this class is as follows:

```
public class LogicalCreateSlotBuilder
    extends AbstractCreateSlotBuilder<ChainedLogicalCreateSlotBuilder>
    implements ChainedLogicalCreateSlotBuilder
```

Constructor

public LogicalCreateSlotBuilder(BaseConnection connection)

Common Methods

Table 7-18 Common methods of LogicalCreateSlotBuilder

Return Value	Method	Description	throws
T	withSlotName(String slotName)	Specifies the name of a replication slot.	-
ChainedLogicalCreateSlotBuilder	withOutputPlugin(String outputPlugin)	Plug-in name. Currently, mppdb_decoding is supported.	-
void	make()	Creates a slot with the specified parameters in the database.	SQLException
ChainedLogicalCreateSlotBuilder	self()	-	-

7.2 ODBC

The ODBC interface is a set of API functions provided to users. This chapter describes its common interfaces. For details on other interfaces, see "ODBC Programmer's Reference" at MSDN (https://msdn.microsoft.com/en-us/library/windows/desktop/ms714177(v=vs.85).aspx).

7.2.1 SQLAllocEnv

In ODBC 3.x, SQLAllocEnv (an ODBC 2.x function) was deprecated and replaced by SQLAllocHandle. For details, see **SQLAllocHandle**.

7.2.2 SQLAllocConnect

In ODBC 3.x, SQLAllocConnect (an ODBC 2.x function) was deprecated and replaced by SQLAllocHandle. For details, see **SQLAllocHandle**.

7.2.3 SQLAllocHandle

Function

SQLAllocHandle is used to allocate environment, connection, statement, or descriptor handles. This function replaces the deprecated ODBC 2.x functions SQLAllocEnv, SQLAllocConnect, and SQLAllocStmt.

Prototype

```
SQLRETURN SQLAllocHandle(SQLSMALLINT   HandleType,
                         SQLHANDLE     InputHandle,
                         SQLHANDLE     *OutputHandlePtr);
```

Parameter

Table 7-19 SQLAllocHandle parameters

Keyword	Parameter Description
HandleType	Type of handle to be allocated by SQLAllocHandle. The value must be one of the following: ● SQL_HANDLE_ENV (environment handle) ● SQL_HANDLE_DBC (connection handle) ● SQL_HANDLE_STMT (statement handle) ● SQL_HANDLE_DESC (descriptor handle) The handle application sequence is: **SQL_HANDLE_ENV** > **SQL_HANDLE_DBC** > **SQL_HANDLE_STMT**. The handle applied later depends on the handle applied prior to it.
InputHandle	Existing handle to use as a context for the new handle being allocated. ● If **HandleType** is set to **SQL_HANDLE_ENV**, this parameter is set to **SQL_NULL_HANDLE**. ● If **HandleType** is set to **SQL_HANDLE_DBC**, this parameter value must be an environment handle. ● If **HandleType** is set to **SQL_HANDLE_STMT** or **SQL_HANDLE_DESC**, this parameter value must be a connection handle.
OutputHandlePtr	**Output parameter**: Pointer to a buffer that stores the returned handle in the newly allocated data structure.

Return Value

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.

- **SQL_INVALID_HANDLE** indicates that invalid handles were called. This value may also be returned by other APIs.

Precautions

If SQLAllocHandle returns **SQL_ERROR** when it is used to allocate a non-environment handle, it sets **OutputHandlePtr** to **SQL_NULL_HDBC**, **SQL_NULL_HSTMT**, or **SQL_NULL_HDESC**. The application can then call **SQLGetDiagRec**, with **HandleType** and **Handle** set to the value of **IntputHandle**, to obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Examples

See **Example: Common Functions and Batch Binding**.

7.2.4 SQLAllocStmt

In ODBC 3.x, SQLAllocStmt was deprecated and replaced by SQLAllocHandle. For details, see **SQLAllocHandle**.

7.2.5 SQLBindCol

Function

SQLBindCol is used to bind columns in a result set to an application data buffer.

Prototype

```
SQLRETURN SQLBindCol(SQLHSTMT       StatementHandle,
                     SQLUSMALLINT   ColumnNumber,
                     SQLSMALLINT    TargetType,
                     SQLPOINTER     TargetValuePtr,
                     SQLLEN      BufferLength,
                     SQLLEN      *StrLen_or_IndPtr);
```

Parameters

Table 7-20 SQLBindCol parameters

Keyword	Parameter Description
StatementHandle	Statement handle.
ColumnNumber	Number of the column to be bound. The column number starts with 0 and increases in ascending order. Column 0 is the bookmark column. If no bookmark column is set, column numbers start with 1.
TargetType	C data type in the buffer.
TargetValuePtr	**Output parameter**: pointer to the buffer bound with the column. The SQLFetch function returns data in the buffer. If **TargetValuePtr** is null, **StrLen_or_IndPtr** is a valid value.

Keyword	Parameter Description
BufferLength	Length of the **TargetValuePtr** buffer in bytes.
StrLen_or_IndPtr	**Output parameter**: pointer to the length or indicator of the buffer. If **StrLen_or_IndPtr** is null, no length or indicator is used.

Return Value

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. This value may also be returned by other APIs.

Precautions

If SQLBindCol returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can call **SQLGetDiagRec**, with **HandleType** and **Handle** set to **SQL_HANDLE_STMT** and **StatementHandle**, respectively, to obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Example

See **Example: Common Functions and Batch Binding**.

7.2.6 SQLBindParameter

Function

SQLBindParameter is used to bind parameter markers in an SQL statement to a buffer.

Prototype

```
SQLRETURN SQLBindParameter(SQLHSTMT       StatementHandle,
                           SQLUSMALLINT   ParameterNumber,
                           SQLSMALLINT    InputOutputType,
                           SQLSMALLINT    ValuetType,
                           SQLSMALLINT    ParameterType,
                           SQLULEN    ColumnSize,
                           SQLSMALLINT    DecimalDigits,
                           SQLPOINTER     ParameterValuePtr,
                           SQLLEN     BufferLength,
                           SQLLEN     *StrLen_or_IndPtr);
```

Parameters

Table 7-21 SQLBindParameter

Keyword	Parameter Description
StatementHandle	Statement handle.
ParameterNumber	Parameter marker number, starting with 1 and increasing in ascending order.
InputOutputType	Input/output type of the parameter.
ValueType	C data type of the parameter.
ParameterType	SQL data type of the parameter.
ColumnSize	Size of the column or expression of the corresponding parameter marker.
DecimalDigits	Decimal digit of the column or the expression of the corresponding parameter marker.
ParameterValuePtr	Pointer to the storage parameter buffer.
BufferLength	Length of the **ParameterValuePtr** buffer in bytes.
StrLen_or_IndPtr	Pointer to the length or indicator of the buffer. If **StrLen_or_IndPtr** is null, no length or indicator is used.

Return Value

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. This value may also be returned by other APIs.

Precautions

If SQLBindParameter returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can call **SQLGetDiagRec**, with **HandleType** and **Handle** set to **SQL_HANDLE_STMT** and **StatementHandle**, respectively, to obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Example

See **Example: Common Functions and Batch Binding**.

7.2.7 SQLColAttribute

Function

SQLColAttribute is used to return the descriptor information about a column in the result set.

Prototype

```
SQLRETURN SQLColAttibute(SQLHSTMT        StatementHandle,
                         SQLUSMALLINT    ColumnNumber,
                         SQLUSMALLINT    FieldIdentifier,
                         SQLPOINTER      CharacterAtrriburePtr,
                         SQLSMALLINT     BufferLength,
                         SQLSMALLINT     *StringLengthPtr,
                         SQLLEN       *NumericAttributePtr);
```

Parameters

Table 7-22 SQLColAttribute parameters

Keyword	Parameter Description
StatementHandle	Statement handle.
ColumnNumber	Column number of the field to be queried, starting with 1 and increasing in ascending order.
FieldIdentifier	Field identifier of **ColumnNumber** in IRD.
CharacterAttribu-tePtr	**Output parameter**: pointer to the buffer that returns the **FieldIdentifier** value.
BufferLength	● **BufferLength** indicates the length of the buffer if **FieldIdentifier** is an ODBC-defined field and **CharacterAttributePtr** points to a string or a binary buffer. ● Ignore this parameter if **FieldIdentifier** is an ODBC-defined field and **CharacterAttributePtr** points to an integer.
StringLengthPtr	**Output parameter**: pointer to a buffer in which the total number of valid bytes (for string data) is stored in ***CharacterAttributePtr**. Ignore the value of **BufferLength** if the data is not a string.
NumericAttributePtr	**Output parameter**: pointer to an integer buffer in which the value of **FieldIdentifier** in the **ColumnNumber** row of the IRD is returned.

Return Value

- **SQL_SUCCESS** indicates that the call succeeded.

- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. This value may also be returned by other APIs.

Precautions

If SQLColAttribute returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can call **SQLGetDiagRec**, with **HandleType** and **Handle** set to **SQL_HANDLE_STMT** and **StatementHandle**, respectively, to obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Example

See **Example: Common Functions and Batch Binding**.

7.2.8 SQLConnect

Function

SQLConnect is used to establish a connection between a driver and a data source. After the connection is established, the connection handle can be used to access all information about the data source, including its application operating status, transaction processing status, and error information.

Prototype

```
SQLRETURN  SQLConnect(SQLHDBC       ConnectionHandle,
                      SQLCHAR       *ServerName,
                      SQLSMALLINT    NameLength1,
                      SQLCHAR       *UserName,
                      SQLSMALLINT    NameLength2,
                      SQLCHAR       *Authentication,
                      SQLSMALLINT    NameLength3);
```

Parameter

Table 7-23 SQLConnect parameters

Keyword	Parameter Description
ConnectionHandle	Connection handle, obtained from SQLAllocHandle.
ServerName	Name of the data source to connect.
NameLength1	Length of **ServerName**.
UserName	Username of the database in the data source.
NameLength2	Length of **UserName**.

Keyword	Parameter Description
Authentication	User password of the database in the data source.
NameLength3	Length of **Authentication**.

Return Value

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. This value may also be returned by other APIs.
- **SQL_STILL_EXECUTING** indicates that the statement is being executed.

Precautions

If SQLConnect returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can call **SQLGetDiagRec**, with **HandleType** and **Handle** set to **SQL_HANDLE_DBC** and **ConnectionHandle**, respectively, to obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Example

See **Example: Common Functions and Batch Binding**.

7.2.9 SQLDisconnect

Function

SQLDisconnect is used to close the connection associated with a database connection handle.

Prototype

```
SQLRETURN SQLDisconnect(SQLHDBC    ConnectionHandle);
```

Parameter

Table 7-24 SQLDisconnect parameters

Keyword	Parameter Description
ConnectionHandle	Connection handle, obtained from SQLAllocHandle.

Return Value

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. This value may also be returned by other APIs.

Precautions

If SQLDisconnect returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can call **SQLGetDiagRec**, with **HandleType** and **Handle** set to **SQL_HANDLE_DBC** and **ConnectionHandle**, respectively, to obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Example

See **Example: Common Functions and Batch Binding**.

7.2.10 SQLExecDirect

Function

SQLExecDirect is used to execute a prepared SQL statement specified in this parameter. This is the fastest method for executing only one SQL statement at a time.

Prototype

```
SQLRETURN SQLExecDirect(SQLHSTMT        StatementHandle,
                        SQLCHAR         *StatementText,
                        SQLINTEGER      TextLength);
```

Parameter

Table 7-25 SQLExecDirect parameters

Keyword	Parameter Description
StatementHandle	Statement handle, obtained from SQLAllocHandle.
StatementText	SQL statement to be executed. One SQL statement can be executed at a time.
TextLength	Length of **StatementText**.

Return Value

- **SQL_SUCCESS** indicates that the call succeeded.

- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_NEED_DATA** indicates that parameters provided before executing the SQL statement are insufficient.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. This value may also be returned by other APIs.
- **SQL_STILL_EXECUTING** indicates that the statement is being executed.
- **SQL_NO_DATA** indicates that the SQL statement does not return a result set.

Precautions

If SQLExecDirect returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can call **SQLGetDiagRec**, with **HandleType** and **Handle** set to **SQL_HANDLE_STMT** and **StatementHandle**, respectively, to obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Example

For details, see **Example: Common Functions and Batch Binding**.

7.2.11 SQLExecute

Function

SQLExecute is used to execute a prepared SQL statement using SQLPrepare. The statement is executed using the current value of any application variables that were bound to parameter markers by SQLBindParameter.

Prototype

```
SQLRETURN SQLExecute(SQLHSTMT    StatementHandle);
```

Parameter

Table 7-26 SQLExecute parameters

Keyword	Parameter Description
StatementHandle	Statement handle to be executed.

Return Value

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.

- **SQL_NEED_DATA** indicates that parameters provided before executing the SQL statement are insufficient.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_NO_DATA** indicates that the SQL statement does not return a result set.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. This value may also be returned by other APIs.
- **SQL_STILL_EXECUTING** indicates that the statement is being executed.

Precautions

If SQLExecute returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can call **SQLGetDiagRec**, with **HandleType** and **Handle** set to **SQL_HANDLE_STMT** and **StatementHandle**, respectively, to obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Example

For details, see **Example: Common Functions and Batch Binding**.

7.2.12 SQLFetch

Function

SQLFetch is used to advance the cursor to the next row of the result set and retrieve any bound columns.

Prototype

```
SQLRETURN SQLFetch(SQLHSTMT    StatementHandle);
```

Parameter

Table 7-27 SQLFetch parameters

Keyword	Parameter Description
StatementHandl e	Statement handle, obtained from SQLAllocHandle.

Return Value

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_NO_DATA** indicates that the SQL statement does not return a result set.

- **SQL_INVALID_HANDLE** indicates that invalid handles were called. This value may also be returned by other APIs.
- **SQL_STILL_EXECUTING** indicates that the statement is being executed.

Precautions

If SQLFetch returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can call **SQLGetDiagRec**, with **HandleType** and **Handle** set to **SQL_HANDLE_STMT** and **StatementHandle**, respectively, to obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Example

For details, see **Example: Common Functions and Batch Binding**.

7.2.13 SQLFreeStmt

In ODBC 3.x, SQLFreeStmt (an ODBC 2.x function) was deprecated and replaced by SQLFreeHandle. For details, see **SQLFreeHandle**.

7.2.14 SQLFreeConnect

In ODBC 3.x, SQLFreeConnect (an ODBC 2.x function) was deprecated and replaced by SQLFreeHandle. For details, see **SQLFreeHandle**.

7.2.15 SQLFreeHandle

Function

SQLFreeHandle is used to release resources associated with a specific environment, connection, or statement handle. It replaces the ODBC 2.x functions: SQLFreeEnv, SQLFreeConnect, and SQLFreeStmt.

Prototype

```
SQLRETURN SQLFreeHandle(SQLSMALLINT    HandleType,
                        SQLHANDLE      Handle);
```

Parameter

Table 7-28 SQLFreeHandle parameters

Keyword	Parameter Description
HandleType	Type of handle to be freed by SQLFreeHandle. The value must be one of the following: ● SQL_HANDLE_ENV ● SQL_HANDLE_DBC ● SQL_HANDLE_STMT ● SQL_HANDLE_DESC If **HandleType** is not one of the preceding values, SQLFreeHandle returns **SQL_INVALID_HANDLE**.
Handle	Name of the handle to be freed.

Return Value

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. This value may also be returned by other APIs.

Precautions

If SQLFreeHandle returns **SQL_ERROR**, the handle is still valid.

Example

For details, see **Example: Common Functions and Batch Binding**.

7.2.16 SQLFreeEnv

In ODBC 3.x, SQLFreeEnv (an ODBC 2.x function) was deprecated and replaced by SQLFreeHandle. For details, see **SQLFreeHandle**.

7.2.17 SQLPrepare

Function

SQLPrepare is used to prepare an SQL statement to be executed.

Note that the prepared statements sent by ODBC do not support the kernel reuse plan. As a result, a new plan needs to be generated for each execution, causing high CPU usage. If services have requirements on plan reuse, you are advised to use the JDBC client.

Prototype

```
SQLRETURN SQLPrepare(SQLHSTMT      StatementHandle,
                     SQLCHAR       *StatementText,
                     SQLINTEGER    TextLength);
```

Parameter

Table 7-29 SQLPrepare parameters

Keyword	Parameter Description
StatementHandle	Statement handle.
StatementText	SQL text string.
TextLength	Length of **StatementText**.

Return Value

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. This value may also be returned by other APIs.
- **SQL_STILL_EXECUTING** indicates that the statement is being executed.

Precautions

If SQLPrepare returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can call **SQLGetDiagRec**, with **HandleType** and **Handle** set to **SQL_HANDLE_STMT** and **StatementHandle**, respectively, to obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Example

See **Examples**.

7.2.18 SQLGetData

Function

SQLGetData is used to retrieve data for a single column in the result set. It can be called for many times to retrieve data of variable lengths.

Prototype

```
SQLRETURN SQLGetData(SQLHSTMT        StatementHandle,
                     SQLUSMALLINT    Col_or_Param_Num,
                     SQLSMALLINT     TargetType,
                     SQLPOINTER      TargetValuePtr,
                     SQLLEN          BufferLength,
                     SQLLEN          *StrLen_or_IndPtr);
```

Parameter

Table 7-30 SQLGetData parameters

Keyword	Parameter Description
StatementHandle	Statement handle, obtained from SQLAllocHandle.
Col_or_Param_Num	Column number for which the data retrieval is requested. The column number starts with 1 and increases in ascending order. The number of the bookmark column is 0.
TargetType	C data type in the TargetValuePtr buffer. If **TargetType** is **SQL_ARD_TYPE**, the driver uses the data type of the **SQL_DESC_CONCISE_TYPE** field in ARD. If **TargetType** is **SQL_C_DEFAULT**, the driver selects a default data type according to the source SQL data type.
TargetValuePtr	**Output parameter**: pointer to the pointer that points to the buffer where the data is located.
BufferLength	Size of the buffer pointed to by **TargetValuePtr**.
StrLen_or_IndPtr	**Output parameter**: pointer to the buffer where the length or identifier value is returned.

Return Value

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_NO_DATA** indicates that the SQL statement does not return a result set.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. This value may also be returned by other APIs.
- **SQL_STILL_EXECUTING** indicates that the statement is being executed.

Precautions

If SQLGetData returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can call **SQLGetDiagRec**, with **HandleType** and **Handle** set to **SQL_HANDLE_STMT** and **StatementHandle**, respectively, to obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Example

For details, see **Example: Common Functions and Batch Binding**.

7.2.19 SQLGetDiagRec

Function

SQLGetDiagRec is used to return the current values of multiple fields in a diagnostic record that contains error, warning, and status information.

Prototype

```
SQLRETURN  SQLGetDiagRec(SQLSMALLINT    HandleType
                         SQLHANDLE      Handle,
                         SQLSMALLINT    RecNumber,
                         SQLCHAR        *SQLState,
                         SQLINTEGER     *NativeErrorPtr,
                         SQLCHAR        *MessageText,
                         SQLSMALLINT    BufferLength
                         SQLSMALLINT    *TextLengthPtr);
```

Parameter

Table 7-31 SQLGetDiagRec parameters

Keyword	Parameter Description
HandleType	Handle-type identifier that describes the type of handle for which diagnostics are desired. The value must be one of the following: ● SQL_HANDLE_ENV ● SQL_HANDLE_DBC ● SQL_HANDLE_STMT ● SQL_HANDLE_DESC
Handle	Handle for the diagnostic data structure. Its type is indicated by **HandleType**. If **HandleType** is set to **SQL_HANDLE_ENV**, **Handle** may indicate a shared or non-shared environment handle.
RecNumber	Status record from which the application seeks information. **RecNumber** starts with 1.
SQLState	**Output parameter**: pointer to a buffer that saves the 5-character **SQLSTATE** code pertaining to **RecNumber**.
NativeErrorPtr	**Output parameter**: pointer to a buffer that saves the native error code.
MessageText	Pointer to a buffer that saves text strings of diagnostic information.
BufferLength	Length of **MessageText**.

Keyword	Parameter Description
TextLengthPtr	**Output parameter**: pointer to the buffer, the total number of bytes in the returned **MessageText**. If the number of bytes available to return is greater than **BufferLength**, then the diagnostics information text in **MessageText** is truncated to **BufferLength** minus the length of the null termination character.

Return Value

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. This value may also be returned by other APIs.

Precautions

SQLGetDiagRec does not release diagnostic records for itself. It uses the following return values to report execution results:

- **SQL_SUCCESS** indicates that the function successfully returns diagnostic information.
- **SQL_SUCCESS_WITH_INFO** indicates that the **MessageText** buffer is too small to hold the requested diagnostic information. No diagnostic records are generated.
- **SQL_INVALID_HANDLE** indicates that the handle indicated by **HandType** and **Handle** is an invalid handle.
- **SQL_ERROR** indicates that the value of **RecNumber** is less than or equal to **0** or the value of **BufferLength** is less than **0**.

If an ODBC function returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can call SQLGetDiagRec to obtain the **SQLSTATE** value. The possible **SQLSTATE** values are listed as follows:

Table 7-32 SQLSTATE values

SQLSATATE	Error	Description
HY000	General error.	An error occurred for which there is no specific SQLSTATE.
HY001	Memory allocation error.	The driver is unable to allocate memory required to support execution or completion of the function.

SQLSATATE	Error	Description
HY008	Operation canceled.	SQLCancel is called to terminate the statement execution, but the StatementHandle function is still called.
HY010	Function sequence error.	The function is called prior to sending data to data parameters or columns being executed.
HY013	Memory management error.	The function fails to be called. The error may be caused by low memory conditions.
HYT01	Connection timeout.	The timeout period expired before the application was able to connect to the data source.
IM001	Function not supported by the driver.	The called function is not supported by the StatementHandle driver.

Examples

See **Example: Common Functions and Batch Binding**.

7.2.20 SQLSetConnectAttr

Function

SQLSetConnectAttr is used to set connection attributes.

Prototype

```
SQLRETURN SQLSetConnectAttr(SQLHDBC       ConnectionHandle
                            SQLINTEGER    Attribute,
                            SQLPOINTER    ValuePtr,
                            SQLINTEGER    StringLength);
```

Parameter

Table 7-33 SQLSetConnectAttr parameters

Keyword	Parameter Description
ConnectionHandle	Connection handle.
Attribute	Attribute to set.

Keyword	Parameter Description
ValuePtr	Pointer to the **Attribute** value. **ValuePtr** depends on the **Attribute** value, and can be a 32-bit unsigned integer value or a null-terminated string. If the **ValuePtr** parameter is a driver-specific value, it may be a signed integer.
StringLength	If **ValuePtr** points to a string or a binary buffer, **StringLength** is the length of ***ValuePtr**. If **ValuePtr** points to an integer, **StringLength** is ignored.

Return Value

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. This value may also be returned by other APIs.

Precautions

If SQLSetConnectAttr returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can call **SQLGetDiagRec**, with **HandleType** and **Handle** set to **SQL_HANDLE_DBC** and **ConnectionHandle**, respectively, to obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Example

For details, see **Example: Common Functions and Batch Binding**.

7.2.21 SQLSetEnvAttr

Function

SQLSetEnvAttr is used to set environment attributes.

Prototype

```
SQLRETURN SQLSetEnvAttr(SQLHENV       EnvironmentHandle
                        SQLINTEGER    Attribute,
                        SQLPOINTER    ValuePtr,
                        SQLINTEGER    StringLength);
```

Parameter

Table 7-34 SQLSetEnvAttr parameters

Keyword	**Parameter Description**
EnvironmentHandle	Environment handle.
Attribute	Environment attribute to be set. The value must be one of the following: ● **SQL_ATTR_ODBC_VERSION**: ODBC version ● **SQL_CONNECTION_POOLING**: connection pool attribute ● **SQL_OUTPUT_NTS**: string type returned by the driver
ValuePtr	Pointer to the **Attribute** value. **ValuePtr** depends on the **Attribute** value, and can be a 32-bit integer value or a null-terminated string.
StringLength	If **ValuePtr** points to a string or a binary buffer, **StringLength** is the length of ***ValuePtr**. If **ValuePtr** points to an integer, **StringLength** is ignored.

Return Value

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. This value may also be returned by other APIs.

Precautions

If SQLSetEnvAttr returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can call **SQLGetDiagRec**, set **HandleType** and **Handle** to **SQL_HANDLE_ENV** and **EnvironmentHandle**, and obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Example

For details, see **Example: Common Functions and Batch Binding**.

7.2.22 SQLSetStmtAttr

Function

SQLSetStmtAttr is used to set attributes related to a statement.

Prototype

```
SQLRETURN SQLSetStmtAttr(SQLHSTMT      StatementHandle
                         SQLINTEGER    Attribute,
                         SQLPOINTER    ValuePtr,
                         SQLINTEGER    StringLength);
```

Parameter

Table 7-35 SQLSetStmtAttr parameters

Keyword	Parameter Description
StatementHandle	Statement handle.
Attribute	Attribute to set.
ValuePtr	Pointer to the **Attribute** value. **ValuePtr** depends on the **Attribute** value, and can be a 32-bit unsigned integer value or a pointer to a null-terminated string, a binary buffer, or a driver-specified value. If the **ValuePtr** parameter is a driver-specific value, it may be a signed integer.
StringLength	If **ValuePtr** points to a string or a binary buffer, **StringLength** is the length of ***ValuePtr**. If **ValuePtr** points to an integer, **StringLength** is ignored.

Return Value

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. This value may also be returned by other APIs.

Precautions

If SQLSetStmtAttr returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can call **SQLGetDiagRec**, with **HandleType** and **Handle** set to **SQL_HANDLE_STMT** and **StatementHandle**, respectively, to obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Example

For details, see **Example: Common Functions and Batch Binding**.

7.2.23 Examples

Code for Common Functions

```
// The following example shows how to obtain data from GaussDB through the ODBC interface.
// DBtest.c (compile with: libodbc.so)
#include <stdlib.h>
#include <stdio.h>
#include <sqlext.h>
#ifdef WIN32
#include <windows.h>
#endif
SQLHENV       V_OD_Env;        // Handle ODBC environment
SQLHSTMT      V_OD_hstmt;      // Handle statement
SQLHDBC       V_OD_hdbc;       // Handle connection
char          typename[100];
SQLINTEGER    value = 100;
SQLINTEGER    V_OD_erg,V_OD_buffer,V_OD_err,V_OD_id;
int main(int argc,char *argv[])
{
      // 1. Allocate an environment handle.
      V_OD_erg = SQLAllocHandle(SQL_HANDLE_ENV,SQL_NULL_HANDLE,&V_OD_Env);
      if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
      {
           printf("Error AllocHandle\n");
           exit(0);
      }
      // 2. Set environment attributes (version information).
      SQLSetEnvAttr(V_OD_Env, SQL_ATTR_ODBC_VERSION, (void*)SQL_OV_ODBC3, 0);
      // 3. Allocate a connection handle.
      V_OD_erg = SQLAllocHandle(SQL_HANDLE_DBC, V_OD_Env, &V_OD_hdbc);
      if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
      {
           SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
           exit(0);
      }
      // 4. Set connection attributes.
      SQLSetConnectAttr(V_OD_hdbc, SQL_ATTR_AUTOCOMMIT, SQL_AUTOCOMMIT_ON, 0);
     // 5. Connect to the data source. userName and password indicate the username and password for
connecting to the database. Set them as needed.
     // If the username and password have been set in the odbc.ini file, you do not need to set userName or
password here, retaining "" for them. However, you are not advised to do so because the username and
password will be disclosed if the permission for odbc.ini is abused.
      V_OD_erg = SQLConnect(V_OD_hdbc, (SQLCHAR*) "gaussdb", SQL_NTS,
                         (SQLCHAR*) "userName", SQL_NTS,  (SQLCHAR*) "password", SQL_NTS);
      if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
      {
          printf("Error SQLConnect %d\n",V_OD_erg);
          SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
          exit(0);
      }
      printf("Connected !\n");
      // 6. Set statement attributes.
      SQLSetStmtAttr(V_OD_hstmt,SQL_ATTR_QUERY_TIMEOUT,(SQLPOINTER *)3,0);
       // 7. Allocate a statement handle.
      SQLAllocHandle(SQL_HANDLE_STMT, V_OD_hdbc, &V_OD_hstmt);
      // 8. Run SQL statements.
      SQLExecDirect(V_OD_hstmt,"drop table IF EXISTS customer_t1",SQL_NTS);
      SQLExecDirect(V_OD_hstmt,"CREATE TABLE customer_t1(c_customer_sk INTEGER, c_customer_name
VARCHAR(32));",SQL_NTS);
      SQLExecDirect(V_OD_hstmt,"insert into customer_t1 values(25,li)",SQL_NTS);
      // 9. Prepare for execution.
      SQLPrepare(V_OD_hstmt,"insert into customer_t1 values(?)",SQL_NTS);
      // 10. Bind parameters.
      SQLBindParameter(V_OD_hstmt,1,SQL_PARAM_INPUT,SQL_C_SLONG,SQL_INTEGER,0,0,
                      &value,0,NULL);
      // 11. Run prepared statements.
      SQLExecute(V_OD_hstmt);
```

```
    SQLExecDirect(V_OD_hstmt,"select id from testtable",SQL_NTS);
    // 12. Obtain attributes of a specific column in the result set.
    SQLColAttribute(V_OD_hstmt,1,SQL_DESC_TYPE,typename,100,NULL,NULL);
    printf("SQLColAtrribute %s\n",typename);
    // 13. Bind the result set.
    SQLBindCol(V_OD_hstmt,1,SQL_C_SLONG, (SQLPOINTER)&V_OD_buffer,150,
            (SQLLEN *)&V_OD_err);
    // 14. Obtain data in the result set by executing SQLFetch.
    V_OD_erg=SQLFetch(V_OD_hstmt);
    // 15. Obtain and return data by executing SQLGetData.
    while(V_OD_erg != SQL_NO_DATA)
    {
        SQLGetData(V_OD_hstmt,1,SQL_C_SLONG,(SQLPOINTER)&V_OD_id,0,NULL);
        printf("SQLGetData ----ID = %d\n",V_OD_id);
        V_OD_erg=SQLFetch(V_OD_hstmt);
    };
    printf("Done !\n");
    // 16. Disconnect data source connections and release handles.
    SQLFreeHandle(SQL_HANDLE_STMT,V_OD_hstmt);
    SQLDisconnect(V_OD_hdbc);
    SQLFreeHandle(SQL_HANDLE_DBC,V_OD_hdbc);
    SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
    return(0);
 }
```

Code for Batch Processing

```
/**********************************************************************
*Set UseBatchProtocol to 1 in the data source and set the database parameter support_batch_bind
*to on.
*The CHECK_ERROR command is used to check and print error information.
*This example is used to interactively obtain the DSN, data volume to be processed, and volume of ignored
data from users, and insert required data into the test_odbc_batch_insert table.
***********************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <sql.h>
#include <sqlext.h>
#include <string.h>

#include "util.c"

void Exec(SQLHDBC hdbc, SQLCHAR* sql)
{
    SQLRETURN retcode;                  // Return status
    SQLHSTMT hstmt = SQL_NULL_HSTMT;    // Statement handle
    SQLCHAR     loginfo[2048];

    // Allocate Statement Handle
    retcode = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
    CHECK_ERROR(retcode, "SQLAllocHandle(SQL_HANDLE_STMT)",
            hstmt, SQL_HANDLE_STMT);

    // Prepare Statement
    retcode = SQLPrepare(hstmt, (SQLCHAR*) sql, SQL_NTS);
    sprintf((char*)loginfo, "SQLPrepare log: %s", (char*)sql);
    CHECK_ERROR(retcode, loginfo, hstmt, SQL_HANDLE_STMT);

    // Execute Statement
    retcode = SQLExecute(hstmt);
    sprintf((char*)loginfo, "SQLExecute stmt log: %s", (char*)sql);
    CHECK_ERROR(retcode, loginfo, hstmt, SQL_HANDLE_STMT);

    // Free Handle
    retcode = SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
    sprintf((char*)loginfo, "SQLFreeHandle stmt log: %s", (char*)sql);
    CHECK_ERROR(retcode, loginfo, hstmt, SQL_HANDLE_STMT);
}
```

```
int main ()
{
    SQLHENV  henv  = SQL_NULL_HENV;
    SQLHDBC  hdbc  = SQL_NULL_HDBC;
    int      batchCount = 1000;
    SQLLEN   rowsCount = 0;
    int      ignoreCount = 0;

    SQLRETURN   retcode;
    SQLCHAR     dsn[1024] = {'\0'};
    SQLCHAR     loginfo[2048];

   // Interactively obtain data source names.
    getStr("Please input your DSN", (char*)dsn, sizeof(dsn), 'N');
   // Interactively obtain the volume of data to be batch processed.
    getInt("batchCount", &batchCount, 'N', 1);
    do
    {
       // Interactively obtain the volume of batch processing data that is not inserted into the database.
        getInt("ignoreCount", &ignoreCount, 'N', 1);
        if (ignoreCount > batchCount)
        {
            printf("ignoreCount(%d) should be less than batchCount(%d)\n", ignoreCount, batchCount);
        }
    }while(ignoreCount > batchCount);

    retcode = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
    CHECK_ERROR(retcode, "SQLAllocHandle(SQL_HANDLE_ENV)",
                henv, SQL_HANDLE_ENV);

    // Set ODBC Verion
    retcode = SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION,
                                        (SQLPOINTER*)SQL_OV_ODBC3, 0);
    CHECK_ERROR(retcode, "SQLSetEnvAttr(SQL_ATTR_ODBC_VERSION)",
                henv, SQL_HANDLE_ENV);

    // Allocate Connection
    retcode = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
    CHECK_ERROR(retcode, "SQLAllocHandle(SQL_HANDLE_DBC)",
                henv, SQL_HANDLE_DBC);

    // Set Login Timeout
    retcode = SQLSetConnectAttr(hdbc, SQL_LOGIN_TIMEOUT, (SQLPOINTER)5, 0);
    CHECK_ERROR(retcode, "SQLSetConnectAttr(SQL_LOGIN_TIMEOUT)",
                hdbc, SQL_HANDLE_DBC);

    // Set Auto Commit
    retcode = SQLSetConnectAttr(hdbc, SQL_ATTR_AUTOCOMMIT,
                                        (SQLPOINTER)(1), 0);
    CHECK_ERROR(retcode, "SQLSetConnectAttr(SQL_ATTR_AUTOCOMMIT)",
                hdbc, SQL_HANDLE_DBC);

    // Connect to DSN
    sprintf(loginfo, "SQLConnect(DSN:%s)", dsn);
    retcode = SQLConnect(hdbc, (SQLCHAR*) dsn, SQL_NTS,
                                (SQLCHAR*) NULL, 0, NULL, 0);
    CHECK_ERROR(retcode, loginfo, hdbc, SQL_HANDLE_DBC);

    // init table info.
    Exec(hdbc, "drop table if exists test_odbc_batch_insert");
    Exec(hdbc, "create table test_odbc_batch_insert(id int primary key, col varchar2(50))");

   // The following code constructs the data to be inserted based on the data volume entered by users:
    {
        SQLRETURN retcode;
        SQLHSTMT hstmtinesrt = SQL_NULL_HSTMT;
        int          i;
        SQLCHAR      *sql = NULL;
        SQLINTEGER   *ids  = NULL;
        SQLCHAR      *cols = NULL;
        SQLLEN       *bufLenIds = NULL;
        SQLLEN       *bufLenCols = NULL;
        SQLUSMALLINT *operptr = NULL;
        SQLUSMALLINT *statusptr = NULL;
        SQLULEN      process = 0;

        // Data is constructed by column. Each column is stored continuously.
        ids = (SQLINTEGER*)malloc(sizeof(ids[0]) * batchCount);
        cols = (SQLCHAR*)malloc(sizeof(cols[0]) * batchCount * 50);
       // Data size in each row for a column
        bufLenIds = (SQLLEN*)malloc(sizeof(bufLenIds[0]) * batchCount);
        bufLenCols = (SQLLEN*)malloc(sizeof(bufLenCols[0]) * batchCount);
       // Whether this row needs to be processed. The value is SQL_PARAM_IGNORE or
SQL_PARAM_PROCEED.
        operptr = (SQLUSMALLINT*)malloc(sizeof(operptr[0]) * batchCount);
        memset(operptr, 0, sizeof(operptr[0]) * batchCount);
       // Processing result of the row
       // Note: In the database, a statement belongs to one transaction. Therefore, data is processed as a
unit. Either all data is inserted successfully or all data fails to be inserted.
        statusptr = (SQLUSMALLINT*)malloc(sizeof(statusptr[0]) * batchCount);
        memset(statusptr, 88, sizeof(statusptr[0]) * batchCount);

        if (NULL == ids || NULL == cols || NULL == bufLenCols || NULL == bufLenIds)
        {
            fprintf(stderr, "FAILED:\tmalloc data memory failed\n");
            goto exit;
        }

        for (int i = 0; i < batchCount; i++)
        {
            ids[i] = i;
            sprintf(cols + 50 * i, "column test value %d", i);
            bufLenIds[i] = sizeof(ids[i]);
            bufLenCols[i] = strlen(cols + 50 * i);
            operptr[i] = (i < ignoreCount) ? SQL_PARAM_IGNORE : SQL_PARAM_PROCEED;
        }

        // Allocate Statement Handle
        retcode = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmtinesrt);
        CHECK_ERROR(retcode, "SQLAllocHandle(SQL_HANDLE_STMT)",
                    hstmtinesrt, SQL_HANDLE_STMT);

        // Prepare Statement
        sql = (SQLCHAR*)"insert into test_odbc_batch_insert values(?, ?)";
        retcode = SQLPrepare(hstmtinesrt, (SQLCHAR*) sql, SQL_NTS);
        sprintf((char*)loginfo, "SQLPrepare log: %s", (char*)sql);
        CHECK_ERROR(retcode, loginfo, hstmtinesrt, SQL_HANDLE_STMT);

        retcode = SQLSetStmtAttr(hstmtinesrt, SQL_ATTR_PARAMSET_SIZE, (SQLPOINTER)batchCount,
sizeof(batchCount));
        CHECK_ERROR(retcode, "SQLSetStmtAttr", hstmtinesrt, SQL_HANDLE_STMT);

        retcode = SQLBindParameter(hstmtinesrt, 1, SQL_PARAM_INPUT, SQL_C_SLONG, SQL_INTEGER,
sizeof(ids[0]), 0,&(ids[0]), 0, bufLenIds);
        CHECK_ERROR(retcode, "SQLBindParameter for id", hstmtinesrt, SQL_HANDLE_STMT);

        retcode = SQLBindParameter(hstmtinesrt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, 50, 50,
cols, 50, bufLenCols);
        CHECK_ERROR(retcode, "SQLBindParameter for cols", hstmtinesrt, SQL_HANDLE_STMT);

        retcode = SQLSetStmtAttr(hstmtinesrt, SQL_ATTR_PARAMS_PROCESSED_PTR, (SQLPOINTER)&process,
sizeof(process));
        CHECK_ERROR(retcode, "SQLSetStmtAttr for SQL_ATTR_PARAMS_PROCESSED_PTR", hstmtinesrt,
SQL_HANDLE_STMT);

        retcode = SQLSetStmtAttr(hstmtinesrt, SQL_ATTR_PARAM_STATUS_PTR, (SQLPOINTER)statusptr,
sizeof(statusptr[0]) * batchCount);
        CHECK_ERROR(retcode, "SQLSetStmtAttr for SQL_ATTR_PARAM_STATUS_PTR", hstmtinesrt,
SQL_HANDLE_STMT);

        retcode = SQLSetStmtAttr(hstmtinesrt, SQL_ATTR_PARAM_OPERATION_PTR, (SQLPOINTER)operptr,
sizeof(operptr[0]) * batchCount);
        CHECK_ERROR(retcode, "SQLSetStmtAttr for SQL_ATTR_PARAM_OPERATION_PTR", hstmtinesrt,
SQL_HANDLE_STMT);

        retcode = SQLExecute(hstmtinesrt);
        sprintf((char*)loginfo, "SQLExecute stmt log: %s", (char*)sql);
        CHECK_ERROR(retcode, loginfo, hstmtinesrt, SQL_HANDLE_STMT);

        retcode = SQLRowCount(hstmtinesrt, &rowsCount);
        CHECK_ERROR(retcode, "SQLRowCount execution", hstmtinesrt, SQL_HANDLE_STMT);

        if (rowsCount != (batchCount - ignoreCount))
        {
            sprintf(loginfo, "(batchCount - ignoreCount)(%d) != rowsCount(%d)", (batchCount - ignoreCount),
rowsCount);
            CHECK_ERROR(SQL_ERROR, loginfo, NULL, SQL_HANDLE_STMT);
        }
        else
        {
            sprintf(loginfo, "(batchCount - ignoreCount)(%d) == rowsCount(%d)", (batchCount - ignoreCount),
rowsCount);
            CHECK_ERROR(SQL_SUCCESS, loginfo, NULL, SQL_HANDLE_STMT);
        }

        // check row number returned
        if (rowsCount != process)
        {
            sprintf(loginfo, "process(%d) != rowsCount(%d)", process, rowsCount);
            CHECK_ERROR(SQL_ERROR, loginfo, NULL, SQL_HANDLE_STMT);
        }
        else
        {
            sprintf(loginfo, "process(%d) == rowsCount(%d)", process, rowsCount);
            CHECK_ERROR(SQL_SUCCESS, loginfo, NULL, SQL_HANDLE_STMT);
        }

        for (int i = 0; i < batchCount; i++)
        {
            if (i < ignoreCount)
            {
                if (statusptr[i] != SQL_PARAM_UNUSED)
                {
                    sprintf(loginfo, "statusptr[%d](%d) != SQL_PARAM_UNUSED", i, statusptr[i]);
                    CHECK_ERROR(SQL_ERROR, loginfo, NULL, SQL_HANDLE_STMT);
                }
            }
            else if (statusptr[i] != SQL_PARAM_SUCCESS)
            {
                sprintf(loginfo, "statusptr[%d](%d) != SQL_PARAM_SUCCESS", i, statusptr[i]);
                CHECK_ERROR(SQL_ERROR, loginfo, NULL, SQL_HANDLE_STMT);
            }
        }

        retcode = SQLFreeHandle(SQL_HANDLE_STMT, hstmtinesrt);
        sprintf((char*)loginfo, "SQLFreeHandle hstmtinesrt");
        CHECK_ERROR(retcode, loginfo, hstmtinesrt, SQL_HANDLE_STMT);
    }


exit:
    printf ("\nComplete.\n");

    // Connection
    if (hdbc != SQL_NULL_HDBC) {
        SQLDisconnect(hdbc);
        SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
    }

    // Environment
    if (henv != SQL_NULL_HENV)
        SQLFreeHandle(SQL_HANDLE_ENV, henv);

    return 0;
}
```

7.3 libpq

7.3.1 Database Connection Control Functions

Database connection control functions control the connections to database servers. An application can connect to multiple servers at a time. For example, a client connects to multiple databases. Each connection is represented by a PGconn object, which is obtained from the function PQconnectdb, PQconnectdbParams, or PQsetdbLogin. Note that these functions will always return a non-null object pointer, unless there is too little memory to allocate the PGconn object. The interface for establishing a connection is stored in the PGconn object. The PQstatus function can be called to check the return value for a successful connection.

7.3.1.1 PQconnectdbParams

Function

PQconnectdbParams is used to establish a new connection with the database server.

Prototype

```
PGconn *PQconnectdbParams(const char * const *keywords,
                          const char * const *values,
                          int expand_dbname);
```

Parameter

Table 7-36 PQconnectdbParams parameters

Keyword	Parameter Description
keywords	An array of strings, each of which is a keyword.
values	Value assigned to each keyword.
expand_dbname	When **expand_dbname** is non-zero, the **dbname** keyword value can be recognized as a connection string. Only **dbname** that first appears is expanded in this way, and any subsequent **dbname** value is treated as a database name.

Return Value

PGconn * points to the object pointer that contains a connection. The memory is applied for by the function internally.

Precautions

This function establishes a new database connection using the parameters taken from two NULL-terminated arrays. Unlike PQsetdbLogin, the parameter set can be extended without changing the function signature. Therefore, use of this function (or its non-blocking analogs PQconnectStartParams and PQconnectPoll) is preferred for new application programming.

Example

For details, see **Example**.

7.3.1.2 PQconnectdb

Function

PQconnectdb is used to establish a new connection with the database server.

Prototype

```
PGconn *PQconnectdb(const char *conninfo);
```

Parameter

Table 7-37 PQconnectdb parameter

Keyword	Parameter Description
conninfo	Connection string. For details about the fields in the string, see **Link Parameters**.

Return Value

PGconn * points to the object pointer that contains a connection. The memory is applied for by the function internally.

Precautions

- This function establishes a new database connection using the parameters taken from the string **conninfo**.
- The input parameter can be empty, indicating that all default parameters can be used. It can contain one or more values separated by spaces or contain a URL.

Example

For details, see **Example**.

7.3.1.3 PQconninfoParse

Function

PQconninfoParse is used to return parsed connection options based on the connection.

Prototype

```
PQconninfoOption* PQconninfoParse(const char* conninfo, char** errmsg);
```

Parameters

Table 7-38

Keyword	Parameter Description
conninfo	Passed string. This parameter can be left empty. In this case, the default value is used. It can contain one or more values separated by spaces or contain a URL.
errmsg	Error information.

Return Value

PQconninfoOption pointers

7.3.1.4 PQconnectStart

Function

PQconnectStart is used to establish a non-blocking connection with the database server.

Prototype

```
PGconn* PQconnectStart(const char* conninfo);
```

Parameters

Table 7-39

Keyword	Parameter Description
conninfo	String of connection information. This parameter can be left empty. In this case, the default value is used. It can contain one or more values separated by spaces or contain a URI.

Return Value

PGconn pointers

7.3.1.5 PQerrorMessage

Function

PQerrorMessage is used to return error information on a connection.

Prototype

```
char* PQerrorMessage(const PGconn* conn);
```

Parameter

Table 7-40

Keyword	Parameter Description
conn	Connection handle.

Return Value

char pointers

Example

For details, see **Example**.

7.3.1.6 PQsetdbLogin

Function

PQsetdbLogin is used to establish a new connection with the database server.

Prototype

```
PGconn *PQsetdbLogin(const char *pghost,
                     const char *pgport,
                     const char *pgoptions,
                     const char *pgtty,
                     const char *dbName,
                     const char *login,
                     const char *pwd);
```

Parameter

Table 7-41 PQsetdbLogin parameters

Keyword	Parameter Description
pghost	Name of the host to be connected. For details, see the **host** field described in **Link Parameters**.
pgport	Port number of the host server. For details, see the **port** field described in **Link Parameters**.
pgoptions	Command-line options to be sent to the server during running. For details, see the **options** field described in **Link Parameters**.
pgtty	This field can be ignored. (Previously, this field declares the output direction of server logs.)
dbName	Name of the database to be connected. For details, see the **dbname** field described in **Link Parameters**.
login	Username for connection. For details, see the **user** field described in **Link Parameters**.
pwd	Password used for authentication during connection. For details, see the **password** field described in **Link Parameters**.

Return Value

PGconn * points to the object pointer that contains a connection. The memory is applied for by the function internally.

Precautions

- This function is the predecessor of PQconnectdb with a fixed set of parameters. When an undefined parameter is called, its default value is used. Write NULL or an empty string for any one of the fixed parameters that is to be defaulted.
- If the **dbName** value contains an = sign or a valid prefix in the connection URL, it is taken as a conninfo string and passed to PQconnectdb, and the remaining parameters are consistent with PQconnectdbParams parameters.

Examples

For details, see **Example**.

7.3.1.7 PQfinish

Function

PQfinish is used to close the connection to the server and release the memory used by the PGconn object.

Prototype

```
void PQfinish(PGconn *conn);
```

Parameter

Table 7-42 PQfinish parameter

Keyword	Parameter Description
conn	Points to the object pointer that contains the connection information.

Precautions

If the server connection attempt fails (as indicated by PQstatus), the application should call PQfinish to release the memory used by the PGconn object. The PGconn pointer must not be used again after PQfinish has been called.

Example

For details, see **Example**.

7.3.1.8 PQreset

Function

PQreset is used to reset the communication port to the server.

Prototype

```
void PQreset(PGconn *conn);
```

Parameter

Table 7-43 PQreset parameter

Keyword	Parameter Description
conn	Points to the object pointer that contains the connection information.

Precautions

This function will close the connection to the server and attempt to establish a new connection to the same server by using all the parameters previously used. This function is applicable to fault recovery after a connection exception occurs.

Example

For details, see **Example**.

7.3.1.9 PQstatus

Function

PQstatus is used to return the connection status.

Prototype

```
ConnStatusType PQstatus(const PGconn *conn);
```

Parameter

Table 7-44 PQ status parameter

Keyword	Parameter Description
conn	Points to the object pointer that contains the connection information.

Return Value

ConnStatusType indicates the connection status. The enumerated values are as follows:

```
CONNECTION_STARTED
Waiting for the connection to be established.

CONNECTION_MADE
Connection succeeded; waiting to send

CONNECTION_AWAITING_RESPONSE
Waiting for a response from the server.

CONNECTION_AUTH_OK
Authentication received; waiting for backend startup to complete.

CONNECTION_SSL_STARTUP
Negotiating SSL encryption.

CONNECTION_SETENV
Negotiating environment-driven parameter settings.

CONNECTION_OK
Normal connection.

CONNECTION_BAD
Failed connection.
```

Precautions

The connection status can be one of the preceding values. After the asynchronous connection procedure is complete, only two of them, **CONNECTION_OK** and **CONNECTION_BAD**, can return. **CONNECTION_OK** indicates that the connection to the database is normal. **CONNECTION_BAD** indicates that the connection attempt fails. Generally, the **CONNECTION_OK** state remains until PQfinish is called. However, a communication failure may cause the connection status to become to **CONNECTION_BAD** before the connection procedure is complete. In this case, the application can attempt to call PQreset to restore the communication.

Example

For details, see **Example**.

7.3.2 Database Statement Execution Functions

After the connection to the database server is successfully established, you can use the functions described in this section to execute SQL queries and commands.

7.3.2.1 PQclear

Function

PQclear is used to release the storage associated with PGresult. Any query result should be released by PQclear when it is no longer needed.

Prototype

```
void PQclear(PGresult *res);
```

Parameters

Table 7-45 PQclear parameter

Keyword	Parameter Description
res	Object pointer that contains the query result.

Precautions

PGresult is not automatically released. That is, it does not disappear when a new query is submitted or even if you close the connection. To delete it, you must call PQclear. Otherwise, memory leakage occurs.

Example

For details, see **Example**.

7.3.2.2 PQexec

Function

PQexec is used to commit a command to the server and wait for the result.

Prototype

```
PGresult *PQexec(PGconn *conn, const char *command);
```

Parameter

Table 7-46 PQexec parameters

Keyword	Parameter Description
conn	Points to the object pointer that contains the connection information.
command	Query string to be executed.

Return Value

PGresult indicates the object pointer that contains the query result.

Precautions

The PQresultStatus function should be called to check the return value for any errors (including the value of a null pointer, in which **PGRES_FATAL_ERROR** will be returned). The PQerrorMessage function can be called to obtain more information about such errors.

NOTICE

The command string can contain multiple SQL commands separated by semicolons (;). Multiple queries sent in a PQexec call are processed in one transaction, unless there are specific BEGIN/COMMIT commands in the query string to divide the string into multiple transactions. Note that the returned PGresult structure describes only the result of the last command executed from the string. If a command fails, the string processing stops and the returned PGresult describes the error condition.

Example

For details, see **Example**.

7.3.2.3 PQexecParams

Function

PQexecParams is used to run a command to bind one or more parameters.

Prototype

```
PGresult* PQexecParams(PGconn* conn,
                       const char* command,
                       int nParams,
                       const Oid* paramTypes,
                       const char* const* paramValues,
                       const int* paramLengths,
                       const int* paramFormats,
                       int resultFormat);
```

Parameter

Table 7-47 PQexecParams parameters

Keyword	Parameter Description
conn	Connection handle.
command	SQL text string.
nParams	Number of parameters to be bound.
paramTypes	Types of parameters to be bound.
paramValues	Values of parameters to be bound.
paramLengths	Parameter lengths.
paramFormats	Parameter formats (text or binary).
resultFormat	Result format (text or binary).

Return Value

PGresult pointers

7.3.2.4 PQexecParamsBatch

Function

PQexecParamsBatch is used to run a command to bind batches of parameters.

Prototype

```
PGresult* PQexecParamsBatch(PGconn* conn,
                            const char* command,
                            int nParams,
                            int nBatch,
                            const Oid* paramTypes,
                            const char* const* paramValues,
                            const int* paramLengths,
                            const int* paramFormats,
                            int resultFormat);
```

Parameter

Table 7-48 PQexecParamsBatch parameters

Keyword	Parameter Description
conn	Connection handle.
command	SQL text string.
nParams	Number of parameters to be bound.
nBatch	Number of batch operations.
paramTypes	Types of parameters to be bound.
paramValues	Values of parameters to be bound.
paramLengths	Parameter lengths.
paramFormats	Parameter formats (text or binary).
resultFormat	Result format (text or binary).

Return Value

PGresult pointers

7.3.2.5 PQexecPrepared

Function

PQexecPrepared is used to send a request to execute a prepared statement with given parameters and wait for the result.

Prototype

```
PGresult* PQexecPrepared(PGconn* conn,
                    const char* stmtName,
                    int nParams,
                    const char* const* paramValues,
                    const int* paramLengths,
                    const int* paramFormats,
                    int resultFormat);
```

Parameter

Table 7-49 PQexecPrepared parameters

Keyword	Parameter Description
conn	Connection handle.
stmtName	*stmt* name, which can be set to "" or NULL to reference an unnamed statement. Otherwise, it must be the name of an existing prepared statement.
nParams	Parameter quantity.
paramValues	Actual values of parameters.
paramLengths	Actual data lengths of parameters.
paramFormats	Parameter formats (text or binary).
resultFormat	Return result format (text or binary).

Return Value

PGresult pointers

7.3.2.6 PQexecPreparedBatch

Function

PQexecPreparedBatch is used to send a request to execute a prepared statement with batches of given parameters and wait for the result.

Prototype

```
PGresult* PQexecPreparedBatch(PGconn* conn,
                              const char* stmtName,
                              int nParams,
                              int nBatchCount,
                              const char* const* paramValues,
                              const int* paramLengths,
                              const int* paramFormats,
                              int resultFormat);
```

Parameter

Table 7-50 PQexecPreparedBatch parameters

Keyword	Parameter Description
conn	Connection handle.

Keyword	Parameter Description
stmtName	*stmt* name, which can be set to "" or NULL to reference an unnamed statement. Otherwise, it must be the name of an existing prepared statement.
nParams	Parameter quantity.
nBatchCount	Number of batches.
paramValues	Actual values of parameters.
paramLengths	Actual data lengths of parameters.
paramFormats	Parameter formats (text or binary).
resultFormat	Return result format (text or binary).

Return Value

PGresult pointers

7.3.2.7 PQfname

Function

PQfname is used to return the column name associated with the given column number. Column numbers start from 0. The caller should not release the result directly. The result will be released when the associated PGresult handle is passed to PQclear.

Prototype

```
char *PQfname(const PGresult *res,
            int column_number);
```

Parameter

Table 7-51 PQfname parameters

Keyword	Parameter Description
res	Operation result handle.
column_number	Number of columns.

Return Value

char pointers

Example

For details, see **Example**.

7.3.2.8 PQgetvalue

Function

PQgetvalue is used to return a single field value of one row of a PGresult. Row and column numbers start from 0. The caller should not release the result directly. The result will be released when the associated PGresult handle is passed to PQclear.

Prototype

```
char *PQgetvalue(const PGresult *res,
                 int row_number,
                 int column_number);
```

Parameter

Table 7-52 PQgetvalue parameters

Keyword	Parameter Description
res	Operation result handle.
row_number	Number of rows.
column_number	Number of columns.

Return Value

For data in text format, the value returned by PQgetvalue is a null-terminated string representation of the field value.

For binary data, the value is a binary representation determined by the typsend and typreceive functions of the data type.

If this field is left blank, an empty string is returned.

Example

For details, see **Example**.

7.3.2.9 PQnfields

Function

PQnfields is used to return the number of columns (fields) in each row of the query result.

Prototype

```
int PQnfields(const PGresult *res);
```

Parameter

Table 7-53 PQnfields parameters

Keyword	Parameter Description
res	Operation result handle.

Return Value

Value of the int type

Example

For details, see **Example**.

7.3.2.10 PQntuples

Function

PQntuples is used to return the number of rows (tuples) in the query result. An overflow may occur if the return value is out of the value range allowed in a 32-bit OS.

Prototype

```
int PQntuples(const PGresult *res);
```

Parameter

Table 7-54 PQntuples parameters

Keyword	Parameter Description
res	Operation result handle.

Return Value

Value of the int type

Examples

For details, see **Example**.

7.3.2.11 PQprepare

Function

PQprepare is used to submit a request to create a prepared statement with given parameters and wait for completion.

Prototype

```
PGresult *PQprepare(PGconn *conn,
                    const char *stmtName,
                    const char *query,
                    int nParams,
                    const Oid *paramTypes);
```

Parameters

Table 7-55 PQprepare parameters

Keyword	Parameter Description
conn	Points to the object pointer that contains the connection information.
stmtName	Name of **stmt** to be executed.
query	Query string to be executed.
nParams	Parameter quantity.
paramTypes	Array of the parameter type.

Return Value

PGresult indicates the object pointer that contains the query result.

Precautions

- PQprepare creates a prepared statement for later execution with PQexecPrepared. This function allows commands to be repeatedly executed, without being parsed and planned each time they are executed. PQprepare is supported only in protocol 3.0 or later. It will fail when protocol 2.0 is used.
- This function creates a prepared statement named **stmtName** from the query string, which must contain an SQL command. **stmtName** can be **""** to create an unnamed statement. In this case, any pre-existing unnamed statement will be automatically replaced. Otherwise, this is an error if the statement name has been defined in the current session. If any parameters are used, they are referred to in the query as $1, $2, and so on. **nParams** is the number of parameters for which types are pre-specified in the array paramTypes[]. (The array pointer can be **NULL** when **nParams** is **0**.) paramTypes[] specifies the data types to be assigned to the parameter symbols by OID. If **paramTypes** is **NULL**, or any element in the array is **0**, the server assigns a data type to the parameter symbol in the same way as it does for an untyped literal string. In

addition, the query can use parameter symbols whose numbers are greater than **nParams**. Data types of these symbols will also be inferred.

NOTICE

You can also execute the **SQLPREPARE** statement to create a prepared statement that is used with PQexecPrepared. Although there is no libpq function of deleting a prepared statement, the **SQL DEALLOCATE** statement can be used for this purpose.

Example

For details, see **Example**.

7.3.2.12 PQresultStatus

Function

PQresultStatus is used to return the result status of a command.

Prototype

```
ExecStatusType PQresultStatus(const PGresult *res);
```

Parameter

Table 7-56 PQresultStatus parameter

Keyword	Parameter Description
res	Object pointer that contains the query result.

Return Value

PQresultStatus indicates the command execution status. The enumerated values are as follows:

```
PQresultStatus can return one of the following values:
PGRES_EMPTY_QUERY
The string sent to the server was empty.

PGRES_COMMAND_OK
A command that does not return data was successfully executed.

PGRES_TUPLES_OK
A query (such as SELECT or SHOW) that returns data was successfully executed.

PGRES_COPY_OUT
Copy Out (from the server) data transfer started.

PGRES_COPY_IN
Copy In (to the server) data transfer started.

PGRES_BAD_RESPONSE
```

```
The response from the server cannot be understood.

PGRES_NONFATAL_ERROR
A non-fatal error (notification or warning) occurred.

PGRES_FATAL_ERROR
A fatal error occurred.

PGRES_COPY_BOTH
Copy In/Out (to and from the server) data transfer started. This state occurs only in streaming replication.

PGRES_SINGLE_TUPLE
PGresult contains a result tuple from the current command. This state occurs in a single-row query.
```

Precautions

- Note that the SELECT command that happens to retrieve zero rows still returns **PGRES_TUPLES_OK**. **PGRES_COMMAND_OK** is used for commands that can never return rows (such as INSERT or UPDATE, without return clauses). The result status **PGRES_EMPTY_QUERY** might indicate a bug in the client software.
- The result status **PGRES_NONFATAL_ERROR** will never be returned directly by PQexec or other query execution functions. Instead, such results will be passed to the notice processor.

Example

For details, see **Example**.

7.3.3 Functions for Asynchronous Command Processing

The PQexec function is adequate for committing commands in common, synchronous applications. However, it has several defects, which may be important to some users:

- PQexec waits for the end of the command, but the application may have other work to do (for example, maintaining a user interface). In this case, PQexec would not want to be blocked to wait for the response.
- As the client application is suspended while waiting for the result, it is difficult for the application to determine whether to cancel the ongoing command.
- PQexec can return only one PGresult structure. If the committed command string contains multiple SQL commands, all the PGresult structures except the last PGresult are discarded by PQexec.
- PQexec always collects the entire result of the command and caches it in a PGresult. Although this mode simplifies the error handling logic for applications, it is impractical for results that contain multiple rows.

Applications that do not want to be restricted by these limitations can use the following functions that PQexec is built from: PQsendQuery and PQgetResult. The functions PQsendQueryParams, PQsendPrepare, and PQsendQueryPrepared can also be used with PQgetResult.

7.3.3.1 PQsendQuery

Function

PQsendQuery is used to commit a command to the server without waiting for the result. If the query is successful, **1** is returned. Otherwise, **0** is returned.

Prototype

```
int PQsendQuery(PGconn *conn, const char *command);
```

Parameter

Table 7-57 PQsendQuery parameters

Keyword	Parameter Description
conn	Points to the object pointer that contains the connection information.
command	Query string to be executed.

Return Value

int indicates the execution result. **1** indicates successful execution and **0** indicates an execution failure. The failure cause is stored in **conn->errorMessage**.

Precautions

After PQsendQuery is successfully called, call PQgetResult one or more times to obtain the results. PQsendQuery cannot be called again (on the same connection) until PQgetResult returns a null pointer, indicating that the command execution is complete.

Examples

For details, see **Example**.

7.3.3.2 PQsendQueryParams

Function

PQsendQueryParams is used to submit a command and separate parameters to the server without waiting for the result.

Prototype

```
int PQsendQueryParams(PGconn *conn,
                      const char *command,
                      int nParams,
                      const Oid *paramTypes,
                      const char * const *paramValues,
                      const int *paramLengths,
                      const int *paramFormats,
                      int resultFormat);
```

Parameter

Table 7-58 PQsendQueryParams parameters

Keyword	Parameter Description
conn	Points to the object pointer that contains the connection information.
command	Query string to be executed.
nParams	Parameter quantity.
paramTypes	Parameter type.
paramValues	Parameter value.
paramLengths	Parameter length.
paramFormats	Parameter format.
resultFormat	Result format.

Return Value

int indicates the execution result. **1** indicates successful execution and **0** indicates an execution failure. The failure cause is stored in **conn->errorMessage**.

Precautions

PQsendQueryParams is equivalent to PQsendQuery. The only difference is that query parameters can be specified separately from the query string. PQsendQueryParams parameters are handled in the same way as PQexecParams parameters. Like PQexecParams, PQsendQueryParams cannot work on connections using protocol v2.0 and it allows only one command in the query string.

Examples

For details, see **Example**.

7.3.3.3 PQsendPrepare

Function

PQsendPrepare is used to send a request to create a prepared statement with given parameters, without waiting for completion.

Prototype

```
int PQsendPrepare(PGconn *conn,
                const char *stmtName,
                const char *query,
                int nParams,
                const Oid *paramTypes);
```

Parameters

Table 7-59 PQsendPrepare parameters

Keyword	Parameter Description
conn	Points to the object pointer that contains the connection information.
stmtName	Name of **stmt** to be executed.
query	Query string to be executed.
nParams	Parameter quantity.
paramTypes	Array of the parameter type.

Return Value

int indicates the execution result. **1** indicates successful execution and **0** indicates an execution failure. The failure cause is stored in **conn->errorMessage**.

Precautions

PQsendPrepare is an asynchronous version of PQprepare. If it can dispatch a request, **1** is returned. Otherwise, **0** is returned. After a successful calling of PQsendPrepare, call PQgetResult to check whether the server successfully created the prepared statement. PQsendPrepare parameters are handled in the same way as PQprepare parameters. Like PQprepare, PQsendPrepare cannot work on connections using protocol 2.0.

Example

For details, see **Example**.

7.3.3.4 PQsendQueryPrepared

Function

PQsendQueryPrepared is used to send a request to execute a prepared statement with given parameters, without waiting for the result.

Prototype

```
int PQsendQueryPrepared(PGconn *conn,
                        const char *stmtName,
                        int nParams,
                        const char * const *paramValues,
                        const int *paramLengths,
                        const int *paramFormats,
                        int resultFormat);
```

Parameters

Table 7-60 PQsendQueryPrepared parameters

Keyword	Parameter Description
conn	Points to the object pointer that contains the connection information.
stmtName	Name of **stmt** to be executed.
nParams	Parameter quantity.
paramValues	Parameter value.
paramLengths	Parameter length.
paramFormats	Parameter format.
resultFormat	Result format.

Return Value

int indicates the execution result. **1** indicates successful execution and **0** indicates an execution failure. The failure cause is stored in **conn->errorMessage**.

Precautions

PQsendQueryPrepared is similar to PQsendQueryParams, but the command to be executed is specified by naming a previously-prepared statement, instead of providing a query string. PQsendQueryPrepared parameters are handled in the same way as PQexecPrepared parameters. Like PQexecPrepared, PQsendQueryPrepared cannot work on connections using protocol 2.0.

Example

For details, see **Example**.

7.3.3.5 PQflush

Function

PQflush is used to try to flush any queued output data to the server.

Prototype

```
int PQflush(PGconn *conn);
```

Parameter

Table 7-61 PQflush parameter

Keyword	Parameter Description
conn	Points to the object pointer that contains the connection information.

Return Value

int indicates the execution result. If the operation is successful (or the send queue is empty), **0** is returned. If the operation fails, **-1** is returned. If all data in the send queue fails to be sent, **1** is returned. (This case occurs only when the connection is non-blocking.) The failure cause is stored in **conn->error_message**.

Precautions

Call PQflush after sending any command or data over a non-blocking connection. If **1** is returned, wait for the socket to become read- or write-ready. If the socket becomes write-ready, call PQflush again. If the socket becomes read-ready, call PQconsumeInput and then call PQflush again. Repeat the operation until the value **0** is returned for PQflush. (It is necessary to check for read-ready and drain the input using PQconsumeInput. This is because the server can block trying to send us data, for example, notification messages, and will not read our data until we read it.) Once PQflush returns **0**, wait for the socket to be read-ready and then read the response as described above.

Example

For details, see **Example**.

7.3.4 Functions for Canceling Queries in Progress

A client application can use the functions described in this section to cancel a command that is still being processed by the server.

7.3.4.1 PQgetCancel

Function

PQgetCancel is used to create a data structure that contains the information required to cancel a command issued through a specific database connection.

Prototype

```
PGcancel *PQgetCancel(PGconn *conn);
```

Parameter

Table 7-62 PQgetCancel parameter

Keyword	Parameter Description
conn	Points to the object pointer that contains the connection information.

Return Value

PGcancel points to the object pointer that contains the cancel information.

Precautions

PQgetCancel creates a PGcancel object for a given PGconn connection object. If the given connection object (**conn**) is NULL or an invalid connection, PQgetCancel will return NULL. The PGcancel object is an opaque structure that cannot be directly accessed by applications. It can be transferred only to PQcancel or PQfreeCancel.

Example

For details, see **Example**.

7.3.4.2 PQfreeCancel

Function

PQfreeCancel is used to release the data structure created by PQgetCancel.

Prototype

```
void PQfreeCancel(PGcancel *cancel);
```

Parameter

Table 7-63 PQfreeCancel parameter

Keyword	Parameter Description
cancel	Points to the object pointer that contains the cancel information.

Precautions

PQfreeCancel releases a data object previously created by PQgetCancel.

Example

For details, see **Example**.

7.3.4.3 PQcancel

Function

PQcancel is used to request the server to abandon processing of the current command.

Prototype

```
int PQcancel(PGcancel *cancel, char *errbuf, int errbufsize);
```

Parameter

Table 7-64 PQcancel parameters

Keyword	Parameter Description
cancel	Points to the object pointer that contains the cancel information.
errbuf	Buffer for storing error information.
errbufsize	Size of the buffer for storing error information.

Return Value

int indicates the execution result. **1** indicates successful execution and **0** indicates an execution failure. The failure cause is stored in **errbuf**.

Precautions

- Successful sending does not guarantee that the request will have any effect. If the cancellation is valid, the current command is terminated early and an error is returned. If the cancellation fails (for example, because the server has processed the command), no result is returned.
- If **errbuf** is a local variable in a signal handler, you can safely call PQcancel from the signal handler. For PQcancel, the PGcancel object is read-only, so it can also be called from a thread that is separate from the thread that is operating the PGconn object.

Example

For details, see **Example**.

7.4 Psycopg

Psycopg APIs are a set of methods provided for users. This section describes some common APIs.

7.4.1 psycopg2.connect()

Function

This method creates a database session and returns a new connection object.

Prototype

```
conn=psycopg2.connect(dbname="test",user="postgres",password="secret",host="127.0.0.1",port="5432")
```

Parameter

Table 7-65 psycopg2.connect parameters

Keyword	Description
dbname	Database name
user	Username
password	Password
host	IP address of the database. The default type is UNIX socket.
port	Connection port number. The default value is **5432**.
sslmode	SSL mode, which is used for SSL connection.
sslcert	Path of the client certificate, which is used for SSL connection.
sslkey	Path of the client key, which is used for SSL connection.
sslrootcert	Path of the root certificate, which is used for SSL connection.
hostaddr	IP address of the database
connect_timeout	Client connection timeout interval
client_encoding	Encoding format of the client
application_name	Value of **application_name**
fallback_application_name	Rollback value of **application_name**

Keyword	Description
keepalives	Specifies whether to enable the TCP connection on the client. The default value is **1**, indicating that the TCP connection is enabled. The value **0** indicates that the TCP connection is disabled. If the UNIX domain socket connection is used, ignore this parameter.
options	Specifies the command line options sent to the server when the connection starts.
keepalives_idle	Describes inactivity before keepalive messages are sent to the server. If keepalive messages are disabled, ignore this parameter.
keepalives_interval	Specifies whether keepalive messages that are not confirmed by the server need to be resent. If keepalive messages are disabled, ignore this parameter.
keepalives_count	Specifies the number of TCP connections that may be lost before the client is disconnected from the server.
replication	Ensures that the connection uses the replication protocol instead of the common protocol.
requiressl	Supports the SSL mode.
sslcompression	Specifies the SSL compression. If this parameter is set to **1**, the data sent through the SSL connection is compressed. If this parameter is set to **0**, compression is disabled. If no SSL connection is established, ignore this parameter.
sslcrl	Specifies the path of the CRL, which is used to check whether the SSL server certificate is available.
requirepeer	Specifies the OS username of the server.

Return Value

Connection object (for connecting to the GaussDB database instance)

Examples

For details, see **Example: Common Operations**.

7.4.2 connection.cursor()

Function

This method returns a new cursor object.

Prototype

```
cursor(name=None, cursor_factory=None, scrollable=None, withhold=False)
```

Parameter

Table 7-66 connection.cursor parameters

Keyword	Description
name	Cursor name. The default value is **None**.
cursor_factory	Creates a non-standard cursor. The default value is **None**.
scrollable	Sets the SCROLL option. The default value is **None**.
withhold	Sets the HOLD option. The default value is **False**.

Return Value

Cursor object (used for cusors that are programmed using Python in the entire database)

Examples

For details, see **Example: Common Operations**.

7.4.3 cursor.execute(query,vars_list)

Function

This method executes the parameterized SQL statements (that is, placeholders instead of SQL literals). The psycopg2 module supports placeholders marked with **%s**.

Prototype

```
curosr.execute(query,vars_list)
```

Parameters

Table 7-67 curosr.execute parameters

Keyword	Description
query	SQL statement to be executed.
vars_list	Variable list, which matches the **%s** placeholder in the query.

Return Value

None

Examples

For details, see **Example: Common Operations**.

7.4.4 curosr.executemany(query,vars_list)

Function

This method executes an SQL command against all parameter sequences or mappings found in the sequence SQL.

Prototype

```
curosr.executemany(query,vars_list)
```

Parameter

Table 7-68 curosr.executemany parameters

Keyword	Description
query	SQL statement that you want to execute.
vars_list	Variable list, which matches the **%s** placeholder in the query.

Return Value

None

Examples

For details, see **Example: Common Operations**.

7.4.5 connection.commit()

Function

This method commits the currently pending transaction to the database.

⚠ CAUTION

By default, Psycopg opens a transaction before executing the first command. If **commit()** is not called, the effect of any data operation will be lost.

Prototype

```
connection.commit()
```

Parameter

None

Return Value

None

Examples

For details, see **Example: Common Operations**.

7.4.6 connection.rollback()

Function

This method rolls back the current pending transaction.

CAUTION

If you close the connection using **close()** but do not commit the change using **commit()**, an implicit rollback will be performed.

Prototype

```
connection.rollback()
```

Parameter

None

Return Value

None

Examples

For details, see **Example: Common Operations**.

7.4.7 cursor.fetchone()

Function

This method extracts the next row of the query result set and returns a tuple.

Prototype

```
cursor.fetchone()
```

Parameter

None

Return Value

A single tuple is the first result in the result set. If no more data is available, **None** is returned.

Examples

For details, see **Example: Common Operations**.

7.4.8 cursor.fetchall()

Function

This method obtains all the (remaining) rows of the query result and returns them as a list of tuples.

Prototype

```
cursor.fetchall()
```

Parameter

None

Return Value

Tuple list, which contains all results of the result set. An empty list is returned when no rows are available.

Examples

For details, see **Example: Common Operations**.

7.4.9 cursor.close()

Function

This method closes the cursor of the current connection.

Prototype

```
cursor.close()
```

Parameter

None

Return Value

None

Examples

For details, see **Example: Common Operations**.

7.4.10 connection.close()

Function

This method closes the database connection.

CAUTION

This method closes the database connection and does not automatically call **commit()**. If you just close the database connection without calling **commit()** first, changes will be lost.

Prototype

```
connection.close()
```

Parameter

None

Return Value

None

Examples

For details, see **Example: Common Operations**.

7.5 Go

7.5.1 sql.Open

The following table describes sql.Open.

Method	Description	Return Value
Open(driverName, dataSourceName string)	Opens a database based on a specified database driver and the dedicated data source of the driver.	*DB and error

For details about the **driverName** and **dataSourceName** parameters, see **Connecting to the Database**.

7.5.2 type DB

The following table describes type DB.

Method	Description	Return Value
(db *DB)Begin()	Starts a transaction. The isolation level of the transaction is determined by the driver.	*Tx and error
(db *DB)BeginTx(ctx context.Context, opts *TxOptions)	Starts a transaction with a specified transaction isolation level. A specified context is used until the transaction is committed or rolled back. If the context is canceled, the SQL package rolls back the transaction.	*Tx and error
(db *DB)Close()	Closes the database and releases all the opened resources.	error
(db *DB)Exec(query string, args ...interface{})	Performs an operation that does not return rows of data.	Result and error
(db *DB)ExecContext(ctx context.Context, query string, args ...interface{})	Performs an operation that does not return rows of data in a specified context.	Result and error
(db *DB)Ping()	Checks whether the database connection is still valid and establishes a connection if necessary.	error
(db *DB)PingContext(ctx context.Context)	Checks whether the database connection is still valid in a specified context and establishes a connection if necessary.	error
(db *DB)Prepare(query string)	Creates a prepared statement for subsequent queries or executions.	*Stmt and error
(db *DB)PrepareContext(ctx context.Context, query string)	Creates a prepared statement for subsequent queries or executions in a specified context.	*Stmt and error
(db *DB)Query(query string, args ...interface{})	Executes a query and returns multiple rows of data.	*Rows and error
(db *DB)QueryContext(ctx context.Context, query string, args ...interface{})	Executes a query and returns multiple rows of data in a specified context.	*Rows and error

(db *DB)QueryRow(query string, args ...interface{})	Executes a query that returns only one row of data.	*Row
(db *DB)QueryRowContext(ctx context.Context, query string, args ...interface{})	Executes a query that returns only one row of data in a specified context.	*Row

Parameter Description

Parameter	Description
ctx	Specified context
query	Executed SQL statement
args	Parameter that needs to be bound to the executed SQL statement. Binding by location and binding by name are supported. For details, see the following example.
opts	Transaction isolation level and transaction access mode. The value of **opts.Isolation** can be **sql.LevelReadUncommitted**, **sql.LevelReadCommitted**, **sql.LevelRepeatableRead**, or **sql.LevelSerializable**. The value of **opts.ReadOnly** can be **true** (read only) or **false** (read and write).

Example:

```
func main() {
    str := "host=127.0.0.1 port=1611 user=testuser password=Gauss_234 dbname=postgres sslmode = disable"
    db, err:= sql.Open("opengauss", str)
    if err != nil {
        log.Fatal(err)
    }
    defer db.Close()

    err = db.Ping()
    if err != nil {
        log.Fatal(err)
    }

    // Binding by location
    _, err := db.Exec("insert into test(id, name) values(:1, :2)", 1, "Zhang San")
    if err != nil {
        log.Fatal(err)
    }

    // Binding by name
    _, err := db.Exec("insert into test(id, name) values(:id, :name)", sql.Named("id", 1), sql.Named ("name,"
"Zhang San"))
    if err != nil {
        log.Fatal(err)
    }
}
```

7.5.3 type Stmt

The following table describes type Stmt.

Method	**Description**	**Return Value**
(s *Stmt)Close()	Closes a specified prepared statement.	error
(s *Stmt)Exec(args ...interface{})	Executes a prepared statement with specified parameters and returns a **Result** value.	Result and error
(s *Stmt)ExecContext(ctx context.Context, args ...interface{})	Executes a prepared statement with specified parameters in a specified context and returns a **Result** value.	Result and error
(s *Stmt)Query(args ...interface{})	Executes a prepared statement with specified parameters and returns ***Rows** as the query result.	*Rows and error
(s *Stmt)QueryContext(ctx context.Context, args ...interface{})	Executes a prepared statement with specified parameters in a specified context and returns ***Rows** as the query result.	*Rows and error
(s *Stmt)QueryRow(args ...interface{})	Executes a prepared statement with specified parameters and returns ***Row** as the result.	*Row
(s *Stmt)QueryRowContext (ctx context.Context, args ...interface{})	Executes a prepared statement with specified parameters in a specified context and returns ***Row** as the result.	*Row

Parameter Description

Parameter	Description
ctx	Specified context
query	Executed SQL statement

args	Parameter that needs to be bound to the executed SQL statement. Binding by location and binding by name are supported. For details, see the example in type DB.

7.5.4 type Tx

The following table describes type Tx.

Method	Description	Return Value
(tx *Tx)Commit()	Commits a transaction.	error
(tx *Tx)Exec(query string, args ...interface{})	Performs an operation that does not return rows of data.	Result and error
(tx *Tx)ExecContext(ctx context.Context, query string, args ...interface{})	Performs an operation that does not return rows of data in a specified context.	Result and error
(tx *Tx)Prepare(query string)	Creates a prepared statement for subsequent queries or executions. The returned statement is executed within a transaction and cannot be used when the transaction is committed or rolled back.	*Stmt and error
(tx *Tx)PrepareContext(ctx context.Context, query string)	Creates a prepared statement for subsequent queries or executions. The returned statement is executed within a transaction and cannot be used when the transaction is committed or rolled back. The specified context will be used in the preparation phase, not in the transaction execution phase. The statement returned by this method will be executed in the transaction context.	*Stmt and error

(tx *Tx)Query(query string, args ...interface{})	Executes a query that returns rows of data.	*Rows and error
(tx *Tx)QueryContext(ctx context.Context, query string, args ...interface{})	Executes a query that returns rows of data in a specified context.	*Rows and error
(tx *Tx)QueryRow(query string, args ...interface{})	Executes a query that returns only one row of data.	*Row
(tx *Tx)QueryRowContext(ctx context.Context, query string, args ...interface{})	Executes a query that returns only one row of data in a specified context.	*Row
(tx *Tx) Rollback()	Rolls back a transaction.	error

(tx *Tx)Stmt(stmt *Stmt)	Returns a transaction-specific prepared statement for an existing statement.
	Example:
	`str, err := db.Prepare("insert into t1 values(:1, :2)")`
	`tx, err := db.Begin()`
	`res, err := tx.Stmt(str).Exec(1, "aaa")`
		*Stmt

(tx *Tx)StmtContext(ctx context.Context, stmt *Stmt)	Returns a transaction-specific prepared statement for an existing statement in a specified context.	*Stmt

Parameter Description

Parameter	Description
ctx	Specified context
query	Executed SQL statement
args	Parameter that needs to be bound to the executed SQL statement. Binding by location and binding by name are supported. For details, see the example in type DB.
stmt	Existing prepared statement, which is generally the prepared statement returned by the **PREPARE** statement

7.5.5 type Rows

The following table describes type Rows.

Method	Description	Return Value
(rs *Rows)Close()	Closes **Rows** to stop the iteration of the data set.	error
(rs *Rows)ColumnTypes()	Returns column information.	[]*ColumnType and error
(rs *Rows)Columns()	Returns the name of each column.	[]string and error
(rs *Rows)Err()	Returns any errors that occur during iteration.	error
(rs *Rows)Next()	Prepares the next data row to be read with the Scan method. If there is an additional result set, **true** is returned. Otherwise, **false** is returned.	bool
(rs *Rows)Scan(dest ...interface{})	Copies the columns of the current iterated row of data to the value specified by **dest**.	error
(rs *Rows)NextResultSet() bool	Specifies whether there is an additional result set.	Bool

Parameter Description

Parameter	Description
dest	The column to be queried needs to be copied to the value specified by this parameter.

7.5.6 type Row

The following table describes type Row.

Method	Description	Return Value
(r *Row)Scan(dest ...interface{})	Copies the columns in the current row of data to the value specified by **dest**.	error
(r *Row)Err()	Returns errors that occur during execution.	error

Parameter Description

Parameter	Description
dest	The column to be queried needs to be copied to the value specified by this parameter.

7.5.7 type ColumnType

The following table describes type ColumnType.

Method	**Description**	**Return Value**
(ci *ColumnType)DatabaseTypeName()	Returns the name of the column-type database system. If an empty string is returned, driver-type names are not supported.	error
(ci *ColumnType)DecimalSize()	Returns the scale and precision of the decimal type. If the value of **ok** is **false**, the specified type is unavailable or not supported.	precision, scale int64, ok bool
(ci *ColumnType)Length()	Returns the length of the data column type. If the value of **ok** is **false**, the specified type does not have a length.	length int64, ok bool
(ci *ColumnType)ScanType()	Returns a Go type that can be used for scanning by using Rows.Scan.	reflect.Type
(ci *ColumnType)Name()	Returns the name of a data column.	string

7.5.8 type Result

The following table describes type Result.

Method	**Description**	**Return Value**
(res Result)RowsAffected()	Returns the number of rows affected by the INSERT, DELETE, UPDATE, SELECT, MOVE, FETCH, and COPY operations.	int64 and error

8 Importing Data

You can use **INSERT**, **COPY**, or **\copy** (a **gsql** meta-command) to import data to the GaussDB database. The methods have different characteristics. For details, see **Table 8-1**.

Table 8-1 Import modes

Mode	Characteristics
INSERT	Insert one or more rows of data, or insert data from a specified table.
COPY	Run the **COPY FROM STDIN** statement to write data into the GaussDB database. Service data does not need to be stored in files when it is written from other databases to the GaussDB database through the CopyManager interface driven by JDBC.
\copy, a **gsql** meta-command	Different from the SQL **COPY** statement, the **\copy** command can read data from or write data into only local files on a **gsql** client. **NOTE** **\copy** applies only to small-scale data import in good format. It does not preprocess invalid characters or provide error tolerance. Therefore, **\copy** cannot be used in scenarios where abnormal data exists. **COPY** is preferred for data import.

8.1 Running the INSERT Statement to Insert Data

Run the **INSERT** statement to write data into the GaussDB database in either of the following ways:

- Use the client tool provided by the GaussDB database to write data into GaussDB.

  For details, see **Inserting Data to Tables**.
- Connect to the database using the JDBC or ODBC driver and run the **INSERT** statement to write data into the GaussDB database.

For details, see **Connecting to a Database**.

You can add, modify, and delete database transactions for the GaussDB database. **INSERT** is the simplest way to write data. It is applicable to scenarios with small data volume and low concurrency.

8.2 Running the COPY FROM STDIN Statement to Import Data

8.2.1 Data Import Using COPY FROM STDIN

Run the **COPY FROM STDIN** statement to import data to GaussDB in either of the following ways:

- Write data into GaussDB by typing. For details, see **COPY**.
- Import data from a file or database to GaussDB through the CopyManager interface driven by JDBC. You can use any parameters in the **COPY** syntax.

8.2.2 Introduction to the CopyManager Class

CopyManager is an API class provided by the JDBC driver in GaussDB. It is used to import data to the GaussDB database in batches.

Inheritance Relationship of CopyManager

The CopyManager class is in the **org.postgresql.copy** package and inherits the java.lang.Object class. The declaration of the class is as follows:

```
public class CopyManager
extends Object
```

Construction Method

public CopyManager(BaseConnection connection)

throws SQLException

Common Methods

Table 8-2 Common methods of CopyManager

Return Value	Method	Description	throws
CopyIn	copyIn(String sql)	-	SQLException
long	copyIn(String sql, InputStream from)	Uses **COPY FROM STDIN** to quickly import data to tables in a database from InputStream.	SQLException,IOException

Return Value	Method	Description	throws
long	copyIn(String sql, InputStream from, int bufferSize)	Uses **COPY FROM STDIN** to quickly import data to tables in a database from InputStream.	SQLException,IOException
long	copyIn(String sql, Reader from)	Uses **COPY FROM STDIN** to quickly import data to tables in a database from Reader.	SQLException,IOException
long	copyIn(String sql, Reader from, int bufferSize)	Uses **COPY FROM STDIN** to quickly import data to tables in a database from Reader.	SQLException,IOException
CopyOut	copyOut(String sql)	-	SQLException
long	copyOut(String sql, OutputStream to)	Sends the result set of **COPY TO STDOUT** from the database to the OutputStream class.	SQLException,IOException
long	copyOut(String sql, Writer to)	Sends the result set of **COPY TO STDOUT** from the database to the Writer class.	SQLException,IOException

8.2.3 Handling Import Errors

Scenarios

Handle errors that occurred during data import.

Querying Error Information

Errors that occur when data is imported are divided into data format errors and non-data format errors.

- Data format errors

When creating a foreign table, specify **LOG INTO** *error_table_name*. Data format errors during data import will be written into the specified table. You can run the following SQL statement to query error details:

```
openGauss=# SELECT * FROM error_table_name;
```

Table 8-3 lists the columns of the *error_table_name* table.

Table 8-3 Columns in the error information table

Column Name	Type	Description
nodeid	integer	ID of the node where an error is reported
begintime	timestamp with time zone	Time when a data format error was reported
filename	character varying	Name of the source data file where a data format error occurs
rownum	bigint	Number of the row where a data format error occurs in a source data file
rawrecord	text	Raw record of a data format error in the source data file
detail	text	Error details

- Non-data format errors

  A non-data format error leads to the failure of an entire data import task. You can locate and troubleshoot a non-data format error based on the error message displayed during data import.

Handling Data Import Errors

Troubleshoot data import errors based on obtained error information and descriptions in the following table.

Table 8-4 Handling data import errors

Error Message	Cause	Solution
missing data for column "r_reason_desc"	1. The number of columns in the source data file is less than that in the foreign table. 2. In a TEXT-format source data file, an escape character (for example, \) leads to delimiter or quote mislocation. Example: The target table contains three columns, and the following data is imported. The escape character (\) converts the delimiter (|) into the value of the second column, causing the value of the third column to lose. `BE|Belgium\\|1`	1. If an error is reported due to missing columns, perform the following operations: ● Add the value of the **r_reason_desc** column to the source data file. ● When creating a foreign table, set the parameter **fill_missing_fields** to **on**. In this way, if the last column of a row in the source data file is missing, it will be set to **NULL** and no error will be reported. 2. Check whether the row where an error is reported contains the escape character (\). If the row contains such a character, you are advised to set the parameter **noescaping** to **true** when creating a foreign table, indicating that the escape character (\) and the characters following it are not escaped.
extra data after last expected column	The number of columns in the source data file is greater than that in the foreign table.	● Delete extra columns from the source data file. ● When creating a foreign table, set the parameter **ignore_extra_data** to **on**. In this way, if the number of columns in the source data file is greater than that in the foreign table, the extra columns at the end of rows will not be imported.

Error Message	Cause	Solution
invalid input syntax for type numeric: "a"	The data type is incorrect.	In the source data file, change the data type of the columns to import. If this error information is displayed, change the data type to **numeric**.
null value in column "staff_id" violates not-null constraint	The not-null constraint is violated.	In the source data file, add values to the specified columns. If this error information is displayed, add values to the **staff_id** column.
duplicate key value violates unique constraint "reg_id_pk"	The unique constraint is violated.	● Delete duplicate rows from the source data file. ● Run the **SELECT** statement with the **DISTINCT** keyword to ensure that all imported rows are unique. `openGauss=# INSERT INTO reasons SELECT DISTINCT * FROM foreign_tpcds_reasons;`
value too long for type character varying(16)	The column length exceeds the upper limit.	In the source data file, change the column length. If this error information is displayed, reduce the column length to no greater than 16 bytes (VARCHAR2).

8.2.4 Example 1: Importing and Exporting Data Through Local Files

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

The sample program is as follows. Load the GaussDB JDBC driver before executing it.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.io.IOException;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.sql.SQLException;
import org.postgresql.copy.CopyManager;
import org.postgresql.core.BaseConnection;

public class Copy{

    public static void main(String[] args)
    {
     String urls = new String("jdbc:postgresql://localhost:8000/postgres"); // URL of the database
```

```
        String username = new String("username");            // Username
        String password = new String("passwd");             // Password
        String tablename = new String("migration_table"); // Table information
        String tablename1 = new String("migration_table_1"); // Table information
        String driver = "org.postgresql.Driver";
        Connection conn = null;

        try {
            Class.forName(driver);
            conn = DriverManager.getConnection(urls, username, password);
        } catch (ClassNotFoundException e) {
             e.printStackTrace(System.out);
        } catch (SQLException e) {
             e.printStackTrace(System.out);
        }

        // Export data from the migration_table table to the d:/data.txt file.
        try {
            copyToFile(conn, "d:/data.txt", "(SELECT * FROM migration_table)");
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        // Import data from the d:/data.txt file to the migration_table_1 table.
        try {
            copyFromFile(conn, "d:/data.txt", tablename1);
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

        // Export data from the migration_table_1 table to the d:/data1.txt file.
        try {
            copyToFile(conn, "d:/data1.txt", tablename1);
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    public static void copyFromFile(Connection connection, String filePath, String tableName)
            throws SQLException, IOException {

        FileInputStream fileInputStream = null;

        try {
            CopyManager copyManager = new CopyManager((BaseConnection)connection);
            fileInputStream = new FileInputStream(filePath);
            copyManager.copyIn("COPY " + tableName + " FROM STDIN ", fileInputStream);
        } finally {
            if (fileInputStream != null) {
                try {
                    fileInputStream.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }
        public static void copyToFile(Connection connection, String filePath, String tableOrQuery)
```

```
            throws SQLException, IOException {

        FileOutputStream fileOutputStream = null;

        try {
            CopyManager copyManager = new CopyManager((BaseConnection)connection);
            fileOutputStream = new FileOutputStream(filePath);
            copyManager.copyOut("COPY " + tableOrQuery + " TO STDOUT", fileOutputStream);
        } finally {
            if (fileOutputStream != null) {
                try {
                    fileOutputStream.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

8.2.5 Example 2: Migrating Data from MY

The following example shows how to use CopyManager to migrate data from MY to GaussDB.

```
import java.io.StringReader;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import org.postgresql.copy.CopyManager;
import org.postgresql.core.BaseConnection;

public class Migration{

    public static void main(String[] args) {
        String url = new String("jdbc:postgresql://localhost:8000/postgres"); // URL of the database
        String user = new String("username");            // GaussDB database user name
        String pass = new String("passwd");             // GaussDB database password
        String tablename = new String("migration_table_1"); // Table information
        String delimiter = new String("|");              // Delimiter
        String encoding = new String("UTF8");            // Character set
        String driver = "org.postgresql.Driver";
        StringBuffer buffer = new StringBuffer();       // Buffer to store formatted data

        try {
            // Obtain the query result set of the source database.
            ResultSet rs = getDataSet();

            // Traverse the result set and obtain records row by row.
            // The values of columns in each record are separated by the specified delimiter and end with a
linefeed, forming strings.
            // Add the strings to the buffer.
            while (rs.next()) {
                buffer.append(rs.getString(1) + delimiter
                        + rs.getString(2) + delimiter
                        + rs.getString(3) + delimiter
                        + rs.getString(4)
                        + "\n");
            }
            rs.close();

            try {
                // Connect to the target database.
                Class.forName(driver);
                Connection conn = DriverManager.getConnection(url, user, pass);
                BaseConnection baseConn = (BaseConnection) conn;
                baseConn.setAutoCommit(false);

                // Initialize the table.
                String sql = "Copy " + tablename + " from STDIN with (DELIMITER " + "'" + delimiter + "'" +","+ "
ENCODING " + "'" + encoding + "')";

                // Commit data in the buffer.
                CopyManager cp = new CopyManager(baseConn);
                StringReader reader = new StringReader(buffer.toString());
                cp.copyIn(sql, reader);
                baseConn.commit();
                reader.close();
                baseConn.close();
            } catch (ClassNotFoundException e) {
                e.printStackTrace(System.out);
            } catch (SQLException e) {
                e.printStackTrace(System.out);
            }

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    //********************************
    // Return the query result set from the source database.
    //*********************************
    private static ResultSet getDataSet() {
        ResultSet rs = null;
        try {
            Class.forName("com.MY.jdbc.Driver").newInstance();
            Connection conn = DriverManager.getConnection("jdbc:MY://10.119.179.227:3306/jack?
useSSL=false&allowPublicKeyRetrieval=true", "jack", "xxxxxxxxx");
            Statement stmt = conn.createStatement();
            rs = stmt.executeQuery("select * from migration_table");
        } catch (SQLException e) {
            e.printStackTrace();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return rs;
    }
}
```

8.3 Using a gsql Meta-Command to Import Data

The GSQL tool provides the **\copy** meta-command to import data.

\copy Command

For the format and description of the **\copy** command, see **Table 8-5**.

Table 8-5 \copy meta-command

Syntax	Remarks
\copy { table [(column_list)] | (query) } { from | to } { filename | stdin | stdout | pstdin | pstdout } [with] [binary] [delimiter [as] 'character'] [null [as] 'string'] [csv [header] [quote [as] 'character'] [escape [as] 'character'] [force quote column_list | *] [force not null column_list]]	You can run this command to import or export data after logging in to a database on any gsql client. Different from the **COPY** statement in SQL, this command performs read/write operations on local files rather than files on database servers. The accessibility and permissions of the local files are restricted to local users. **NOTE** **\copy** applies only to small-scale data import in good format. It does not preprocess invalid characters or provide error tolerance. Therefore, **\copy** cannot be used in scenarios where abnormal data exists. **COPY** is preferred for data import.

Parameter Description

- table

  Specifies the name (possibly schema-qualified) of an existing table.

  Value range: an existing table name
- column_list

  Specifies an optional list of columns to be copied.

  Value range: any field in the table. If no column list is specified, all columns of the table will be copied.
- query

  Specifies that the results are to be copied.

  Value range: a **SELECT** or **VALUES** command in parentheses
- filename

  Specifies the absolute path of a file. To run the **COPY** command, the user must have the write permission for this path.
- stdin

  Specifies that input comes from the standard input.
- stdout

  Specifies that output goes to the standard output.
- pstdin

  Specifies that input comes from the gsql client.
- pstout
- Specifies that output goes to the gsql client.
- binary

  Specifies that data is stored and read in binary mode instead of text mode. In binary mode, you cannot declare **DELIMITER**, **NULL**, or **CSV**. After **binary** is specified, CSV, FIXED, and TEXT cannot be specified through **option** or **copy_option**.

- delimiter [as] 'character'

  Specifies the character that separates columns within each row (line) of the file.

  NOTE

  - The value of **delimiter** cannot be **\r** or **\n**.
  - A delimiter cannot be the same as the null value. The delimiter for the CSV format cannot be same as the **quote** value.
  - The delimiter of TEXT data cannot contain any of the following characters: \.abcdefghijklmnopqrstuvwxyz0123456789.
  - The data length of a single row should be less than 1 GB. A row that has many columns using long delimiters cannot contain much valid data.
  - You are advised to use multi-character delimiters or invisible delimiters. For example, you can use multi-characters (such as $^&) and invisible characters (such as 0x07, 0x08, and 0x1b).

  Value range: a multi-character delimiter within 10 bytes

  Default value:

  – A tab character in text format
  – A comma (,) in CSV format
  – No delimiter in FIXED format

- null [as] 'string'

  Specifies the string that represents a null value.

  Value range:

  – A null value cannot be **\r** or **\n**. The maximum length is 100 characters.
  – A null value cannot be the same as the **delimiter** or **quote** value.

  Default value:

  – The default value for the CSV format is an empty string without quotation marks.
  – The default value for the TEXT format is **\N**.

- header

  Specifies whether a file contains a header with the names of each column in the file. **header** is available only for CSV and FIXED files.

  When data is imported, if **header** is **on**, the first row of the data file will be identified as the header and ignored. If **header** is **off**, the first row will be identified as a data row.

  When data is exported, if header is **on**, **fileheader** must be specified. **fileheader** specifies the content in the header. If **header** is **off**, an exported file does not contain a header.

  Value range: **true/on** and **false/off**

  Default value: false

- quote [as] 'character'

  Specifies a quoted character string for a CSV file.

  Default value: ""

NOTE

- The value of **quote** cannot be the same as that of the **delimiter** or null parameter.
- The value of **quote** must be a single-byte character.
- Invisible characters are recommended, such as 0x07, 0x08, and 0x1b.

- escape [as] 'character'

  Specifies an escape character for a CSV file. The value must be a single-byte character.

  Default value: "" If the value is the same as that of **quote**, it will be replaced by **\0**.

- force quote column_list | *

  In **CSV COPY TO** mode, forces quotation marks to be used for all non-null values in each specified column. Null values are not quoted.

  Value range: an existing column name

- force not null column_list

  Assigns a value to a specified column in **CSV COPY FROM** mode.

  Value range: an existing column name

Examples

1. Create a target table **a**.
   ```
   openGauss=# CREATE TABLE a(a int);
   ```
2. Import data.
   a. Copy data from **stdin** to table **a**.
   ```
   openGauss=# \copy a from stdin;
   ```
   When the **>>** characters are displayed, enter data. To end your input, enter a backslash and a period (\.).
   ```
   Enter data to be copied followed by a newline.
   End with a backslash and a period on a line by itself.
   >> 1
   >> 2
   >> \.
   ```
   Query data imported to table **a**.
   ```
   openGauss=# SELECT * FROM a;
    a
   ---
    1
    2
   (2 rows)
   ```
   b. Copy data from a local file to table **a**. The following assumes that the local file is **/home/omm/2.csv**.
   - Commas (,) are used as delimiters.
   - If the number of columns defined in a source data file is greater than that in a foreign table, extra columns will be ignored during import.
   ```
   openGauss=# \copy a FROM '/home/omm/2.csv' WITH (delimiter',',IGNORE_EXTRA_DATA 'on');
   ```

8.4 Updating Data in a Table

8.4.1 Updating a Table by Using DML Statements

In GaussDB, you can update a table by running DML statements.

Procedure

There is a table named **customer_t** and the table structure is as follows:

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

You can run the following DML statements to update data in the table.

- Run the **INSERT** statement to insert data into the table.
  - Insert a row to the **customer_t** table.
    ```
    openGauss=# INSERT INTO customer_t (c_customer_sk, c_customer_id,
    c_first_name,c_last_name) VALUES (3769, 5, 'Grace','White');
    ```
  - Insert multiple rows to the **customer_t** table.
    ```
    openGauss=# INSERT INTO customer_t (c_customer_sk, c_customer_id,
    c_first_name,c_last_name) VALUES
    (6885, 1, 'Joes', 'Hunter'),
    (4321, 2, 'Lily','Carter'),
    (9527, 3, 'James', 'Cook'),
    (9500, 4, 'Lucy', 'Baker');
    ```
    For details on how to use **INSERT**, see **Inserting Data to Tables**.
- Run the **UPDATE** statement to update data in the table. Change the value of the **c_customer_id** column to **0**.
  ```
  openGauss=# UPDATE customer_t SET c_customer_id = 0;
  ```
  For details on how to use **UPDATE**, see **UPDATE**.
- Run the **DELETE** statement to delete rows from the table.

  You can use the **WHERE** clause to specify the rows whose data is to delete. If you do not specify it, all rows in the table are deleted and only the data structure is retained.
  ```
  openGauss=# DELETE FROM customer_t WHERE c_last_name = 'Baker';
  ```
  For details on how to use **DELETE**, see **DELETE**.
- Run the **TRUNCATE** statement to delete all rows from the table.
  ```
  openGauss=# TRUNCATE TABLE customer_t;
  ```
  For details on how to use **TRUNCATE**, see **TRUNCATE**.

  The **DELETE** statement deletes a row of data each time whereas the **TRUNCATE** statement deletes data by releasing the data page stored in the table. Therefore, data can be deleted more quickly by using **TRUNCATE** than using **DELETE**.

  DELETE deletes table data but does not release table storage space. **TRUNCATE** deletes table data and releases table storage space.

8.4.2 Updating and Inserting Data by Using the MERGE INTO Statement

To add all or a large amount of data in a table to an existing table, you can run the **MERGE INTO** statement in GaussDB to merge the two tables so that data can be quickly added to the existing table.

The **MERGE INTO** statement matches data in a source table with that in a target table based on a join condition. If data matches, **UPDATE** will be executed on the target table. Otherwise, **INSERT** will be executed. This statement is a convenient way to combine multiple operations and avoids multiple **INSERT** or **UPDATE** statements.

Prerequisites

You have the **INSERT** and **UPDATE** permissions for the target table and the **SELECT** permission for the source table.

Procedure

Step 1 Create a source table named **products** and insert data.

```
openGauss=# CREATE TABLE products
( product_id INTEGER,
  product_name VARCHAR2(60),
  category VARCHAR2(60)
);

openGauss=# INSERT INTO products VALUES
(1502, 'olympus camera', 'electrncs'),
(1601, 'lamaze', 'toys'),
(1666, 'harry potter', 'toys'),
(1700, 'wait interface', 'books');
```

Step 2 Create a target table named **newproducts** and insert data.

```
openGauss=# CREATE TABLE newproducts
( product_id INTEGER,
  product_name VARCHAR2(60),
  category VARCHAR2(60)
);

openGauss=# INSERT INTO newproducts VALUES
(1501, 'vivitar 35mm', 'electrncs'),
(1502, 'olympus ', 'electrncs'),
(1600, 'play gym', 'toys'),
(1601, 'lamaze', 'toys'),
(1666, 'harry potter', 'dvd');
```

Step 3 Run the **MERGE INTO** statement to merge data in the source table **products** into the target table **newproducts**.

```
openGauss=# MERGE INTO newproducts np
USING products p
ON (np.product_id = p.product_id )
WHEN MATCHED THEN
  UPDATE SET np.product_name = p.product_name, np.category = p.category
WHEN NOT MATCHED THEN
  INSERT VALUES (p.product_id, p.product_name, p.category) ;
```

For details on parameters in the statement, see **Table 8-6**. For more information, see **MERGE INTO**.

Table 8-6 Parameters in the MERGE INTO statement

Parameter	Description	Example Value
INTO clause	Specifies a target table that is to be updated or has data to be inserted. A table alias is supported.	Value: **newproducts np** The table name is **newproducts** and the alias is **np**.
USING clause	Specifies a source table. A table alias is supported.	Value: **products p** The table name is **products** and the alias is **p**.
ON clause	Specifies a join condition between a target table and a source table. Columns in the join condition cannot be updated.	Value: **np.product_id = p.product_id** The join condition is that the **product_id** column in the target table **newproducts** has equivalent values as the **product_id** column in the source table **products**.
WHEN MATCHED clause	Performs **UPDATE** if data in the source table matches that in the target table based on the condition. ● Only one **WHEN MATCHED** clause can be specified. ● The **WHEN MATCHED** clause can be omitted. If it is omitted, no operation will be performed on the rows that meet the condition in the **ON** clause.	Value: **WHEN MATCHED THEN UPDATE SET np.product_name = p.product_name**, **np.category = p.category** When the condition in the **ON** clause is met, the values of the **product_name** and **category** columns in the target table **newproducts** are replaced with the values in the corresponding columns in the source table **products**.

Parameter	Description	Example Value
WHEN NOT MATCHED clause	Performs **INSERT** if data in the source table does not match that in the target table based on the condition. ● Only one **WHEN NOT MATCHED** clause can be specified. ● The **WHEN NOT MATCHED** clause can be omitted. ● An **INSERT** clause can contain only one **VALUES**. ● The **WHEN MATCHED** and **WHEN NOT MATCHED** clauses can be exchanged in sequence. One of them can be omitted, but they cannot be omitted at the same time.	Value: **WHEN NOT MATCHED THEN** **INSERT VALUES (p.product_id, p.product_name, p.category)** Insert rows in the source table **products** that do not meet the condition in the **ON** clause into the target table **newproducts**.

Step 4 Query the target table **newproducts** after the merge.

```
openGauss=# SELECT * FROM newproducts;
```

The command output is as follows:

```
 product_id |  product_name  | category
------------+----------------+-----------
       1501 | vivitar 35mm   | electrncs
       1502 | olympus camera | electrncs
       1666 | harry potter   | toys
       1600 | play gym       | toys
       1601 | lamaze         | toys
       1700 | wait interface | books
(6 rows)
```

----End

8.5 Deep Copy

After data is imported, you can perform a deep copy to modify a partition key, change a row-store table to a column-store table, or add a partial cluster key. A deep copy re-creates a table and batch inserts data into the table.

GaussDB provides three deep copy methods.

8.5.1 Performing a Deep Copy by Using the CREATE TABLE Statement

Run the **CREATE TABLE** statement to create a copy of the original table, batch insert data of the original table into the copy, and rename the copy to the name of the original table.

When creating the copy, you can specify table and column attributes, such as the primary key.

Procedure

Perform the following operations to carry out a deep copy for the **customer_t** table:

Step 1 Run the **CREATE TABLE** statement to create the copy **customer_t_copy** of the **customer_t** table.

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

Step 2 Run the **INSERT INTO...SELECT** statement to batch insert data of the original table into the copy.

```
openGauss=# INSERT INTO customer_t_copy (SELECT * FROM customer_t);
```

Step 3 Delete the original table.

```
openGauss=# DROP TABLE customer_t;
```

Step 4 Run the **ALTER TABLE** statement to rename the copy to the name of the original table.

```
openGauss=# ALTER TABLE customer_t_copy RENAME TO customer_t;
```

----End

8.5.2 Performing a Deep Copy by Using the CREATE TABLE LIKE Statement

Run the **CREATE TABLE LIKE** statement to create a copy of the original table, batch insert data of the original table into the copy, and rename the copy to the name of the original table. This method does not inherit the primary key attributes of the original table. You can use the **ALTER TABLE** statement to add them.

Procedure

Step 1 Run the **CREATE TABLE LIKE** statement to create the copy **customer_t_copy** of the **customer_t** table.

```
openGauss=# CREATE TABLE customer_t_copy (LIKE customer_t);
```

Step 2 Run the **INSERT INTO...SELECT** statement to batch insert data of the original table into the copy.

```
openGauss=# INSERT INTO customer_t_copy (SELECT * FROM customer_t);
```

Step 3 Delete the original table.

```
openGauss=# DROP TABLE customer_t;
```

Step 4 Run the **ALTER TABLE** statement to rename the copy to the name of the original table.

```
openGauss=# ALTER TABLE customer_t_copy RENAME TO customer_t;
```

----End

8.5.3 Performing a Deep Copy by Creating a Temporary Table and Truncating the Original Table

Run the **CREATE TEMP TABLE ... AS** statement to create a temporary table for the original table, truncate the original table, and batch insert data of the temporary data into the original table.

When creating the temporary table, retain the primary key attributes of the original table. This method is recommended if the original table has dependency items.

Procedure

Step 1 Run the **CREATE TEMP TABLE AS** statement to create a temporary table **customer_t_temp** for the **customer_t** table.

```
openGauss=# CREATE TEMP TABLE customer_t_temp AS SELECT * FROM customer_t;
```

NOTE

- Compared with the use of permanent tables, the use of temporary tables can improve performance but may incur data loss. A temporary table is automatically deleted at the end of the session where it is located. If data loss is unacceptable, use a permanent table.
- Temporary tables and common tables are stored in the same location. You can also specify a tablespace to store temporary tables. If too many local temporary tables are used, the system catalog may bloat, but the overall impact is acceptable.

Step 2 Truncate the original table **customer_t**.

```
openGauss=# TRUNCATE customer_t;
```

Step 3 Run the **INSERT INTO...SELECT** statement to batch insert data of the temporary table into the original table.

```
openGauss=# INSERT INTO customer_t (SELECT * FROM customer_t_temp);
```

Step 4 Delete the temporary table **customer_t_temp**.

```
openGauss=# DROP TABLE customer_t_temp;
```

----End

8.6 Analyzing Tables

The execution plan generator needs to use table statistics to generate the most effective query execution plan to improve query performance. After data is imported, you are advised to run the **ANALYZE** statement to update table statistics. The statistics are stored in the system catalog **PG_STATISTIC**.

Analyzing Tables

ANALYZE supports row-store and column-store tables. **ANALYZE** can also collect statistics about specified columns of a local table. For details on **ANALYZE**, see **ANALYZE | ANALYSE**.

Step 1 Update table statistics.

Take the **product_info** table as an example. The **ANALYZE** command is as follows:

```
openGauss=# ANALYZE product_info;
ANALYZE
```

----End

Automatically Analyzing Tables

GaussDB provides the GUC parameter **autovacuum** to specify whether to enable the autovacuum function of the database.

If **autovacuum** is set to **on**, the system will start the autovacuum thread to automatically analyze tables when the data volume in the table reaches the threshold. This is the autoanalyze function.

- For an empty table, when the number of rows inserted to it is greater than 50, **ANALYZE** is automatically triggered.
- For a table containing data, the threshold is 50 + 10% x **reltuples**, where **reltuples** indicates the total number of rows in the table.

The autovacuum function also depends on the following two GUC parameters in addition to **autovacuum**:

- **track_counts**: This parameter must be set to **on** to enable statistics collection about the database.
- **autovacuum_max_workers**: This parameter must be set to a value greater than **0** to specify the maximum number of concurrent autovacuum threads.

NOTICE

- The autoanalyze function supports the default sampling mode but not percentage sampling.
- The autoanalyze function does not collect multi-column statistics, which only supports percentage sampling.
- The autoanalyze function supports row-store and column-store tables and does not support foreign tables, temporary tables, unlogged tables, and TOAST tables.

8.7 Doing VACUUM to a Table

If a large number of rows were updated or deleted during import, run **VACUUM FULL** before **ANALYZE**. A large number of UPDATE and DELETE operations generate huge disk page fragments, which reduces query efficiency. **VACUUM FULL** can restore disk page fragments and return them to the OS.

Step 1 Run the **VACUUM FULL** statement.

Do **VACUUM FULL** to the **product_info** table.

```
openGauss=# VACUUM FULL product_info
VACUUM
```

----End

8.8 Managing Concurrent Write Operations

8.8.1 Transaction Isolation

GaussDB manages transactions based on MVCC and two-phase locks, avoiding conflicts between read and write operations. SELECT is a read-only operation, whereas UPDATE and DELETE are read/write operations.

- There is no conflict between read/write and read-only operations, or between read/write operations. Each concurrent transaction creates a snapshot when it starts. Concurrent transactions cannot detect updates made by each other.
  - At the **READ COMMITTED** level, if transaction T1 is committed, transaction T2 can see changes made by T1.
  - At the **REPEATABLE READ** level, if T2 starts before T1 is committed, T2 will not see changes made by T1 even after T1 is committed. The query results in a transaction are consistent and unaffected by other transactions.
- Read/Write operations use row-level locks. Different transactions can concurrently update the same table but not the same row. A row update transaction will start only after the previous one is committed.
  - **READ COMMITTED**: At this level, a transaction can access only committed data. This is the default level.
  - **REPEATABLE READ**: Only data committed before transaction start is read. Uncommitted data or data committed in other concurrent transactions cannot be read.

8.8.2 Write and Read/Write Operations

Statements for write-only and read/write operations are as follows:

- **INSERT**, used to insert one or more rows of data into a table
- **UPDATE**, used to modify existing data in a table
- **DELETE**, used to delete existing data from a table
- **COPY**, used to import data

INSERT and COPY are write-only operations. Only one of them can be performed at a time. If INSERT or COPY of transaction T1 locks a table, INSERT or COPY of transaction T2 needs to wait until T1 unlocks the table.

UPDATE and DELETE operations are read/write operations. They need to query for the target rows before modifying data. Concurrent transactions cannot see changes made by each other, and UPDATE and DELETE operations read snapshots of data committed before their transactions start. Write operations use row-level locks. If T2 starts after T1 and is to update the same row as T1 does, T2 waits for

T1 to finish update. If T1 is not complete within the specified timeout duration, T2 will time out. If T1 and T2 update different rows in a table, they can be concurrently executed.

8.8.3 Potential Deadlocks During Concurrent Write

Whenever transactions involve updates of more than one table, there is always the possibility that concurrently running transactions become deadlocked when they both try to write to the same set of tables. A transaction releases all of its locks at once when it either commits or rolls back; it does not relinquish locks one at a time. For example, transactions T1 and T2 start at roughly the same time.

- If T1 starts writing to table A and T2 starts writing to table B, both transactions can proceed without conflict. However, if T1 finishes writing to table A and needs to start writing to the same rows as T2 does in table B, it will not be able to proceed because T2 still holds the lock on B. Conversely, if T2 finishes writing to table B and needs to start writing to the same rows as T1 does in table A, it will not be able to proceed either because T1 still holds the lock on A. In this case, a deadlock occurs. If T1 is committed and releases the lock within the lock timeout duration, subsequent update can proceed. If a lock times out, an error is reported and the corresponding transaction exits.
- If T1 updates rows 1 to 5 and T2 updates rows 6 to 10 in the same table, the two transactions do not conflict. However, if T1 finishes the update and proceeds to update rows 6 to 10, and T2 proceeds to update rows 1 to 5, neither of them can continue. If either of the transactions is committed and releases the lock within the lock timeout duration, subsequent update can proceed. If a lock times out, an error is reported and the corresponding transaction exits.

8.8.4 Concurrent Write Examples

This section uses the **test** table as an example to describe how to perform concurrent **INSERT** and **DELETE** in the same table, concurrent **INSERT** in the same table, concurrent **UPDATE** in the same table, and concurrent import and queries.

```
CREATE TABLE test(id int, name char(50), address varchar(255));
```

8.8.4.1 Concurrent INSERT and DELETE in the Same Table

Transaction T1:

```
START TRANSACTION;
INSERT INTO test VALUES(1,'test1','test123');
COMMIT;
```

Transaction T2:

```
START TRANSACTION;
DELETE test WHERE NAME='test1';
COMMIT;
```

Scenario 1:

T1 is started but not committed. At this time, T2 is started. After **INSERT** of T1 is complete, **DELETE** of T2 is performed. In this case, **DELETE 0** is displayed, because T1 is not committed and T2 cannot see the data inserted by T1.

Scenario 2:

- **READ COMMITTED** level

  T1 is started but not committed. At this time, T2 is started. After **INSERT** of T1 is complete, T1 is committed and **DELETE** of T2 is executed. In this case, **DELETE 1** is displayed, because T2 can see the data inserted by T1.
- **REPEATABLE READ** level

  T1 is started but not committed. At this time, T2 is started. After **INSERT** of T1 is complete, T1 is committed and **DELETE** of T2 is executed. In this case, **DELETE 0** is displayed, because the data obtained in queries is consistent in a transaction.

8.8.4.2 Concurrent INSERT in the Same table

Transaction T1:

```
START TRANSACTION;
INSERT INTO test VALUES(2,'test2','test123');
COMMIT;
```

Transaction T2:

```
START TRANSACTION;
INSERT INTO test VALUES(3,'test3','test123');
COMMIT;
```

Scenario 1:

T1 is started but not committed. At this time, T2 is started. After **INSERT** of T1 is complete, **INSERT** of T2 is executed and succeeds. At the **READ COMMITTED** and **REPEATABLE READ** levels, the **SELECT** statement of T1 cannot see data inserted by T2, and a query in T2 cannot see data inserted by T1.

Scenario 2:

- **READ COMMITTED** level

  T1 is started but not committed. At this time, T2 is started. After **INSERT** of T1 is complete, T1 is committed. In T2, a query executed after **INSERT** can see the data inserted by T1.
- **REPEATABLE READ** level

  T1 is started but not committed. At this time, T2 is started. After **INSERT** of T1 is complete, T1 is committed. In T2, a query executed after **INSERT** cannot see the data inserted by T1.

8.8.4.3 Concurrent UPDATE in the Same Table

Transaction T1:

```
START TRANSACTION;
UPDATE test SET address='test1234' WHERE name='test1';
COMMIT;
```

Transaction T2:

```
START TRANSACTION;
UPDATE test SET address='test1234' WHERE name='test2';
COMMIT;
```

Transaction T3:

```
START TRANSACTION;
UPDATE test SET address='test1234' WHERE name='test1';
COMMIT;
```

Scenario 1:

T1 is started but not committed. At this time, T2 is started. **UPDATE** of T1 and then T2 starts, and both of them succeed. This is because the **UPDATE** operations use row-level locks and do not conflict when they update different rows.

Scenario 2:

T1 is started but not committed. At this time, T3 is started. **UPDATE** of T1 and then T3 starts, and **UPDATE** of T1 succeeds. **UPDATE** of T3 times out. This is because T1 and T3 update the same row and the lock is held by T1 at the time of the update.

8.8.4.4 Concurrent Data Import and Queries

Transaction T1:

```
START TRANSACTION;
COPY test FROM '...';
COMMIT;
```

Transaction T2:

```
START TRANSACTION;
SELECT * FROM test;
COMMIT;
```

Scenario 1:

T1 is started but not committed. At this time, T2 is started. **COPY** of T1 and then **SELECT** of T2 starts, and both of them succeed. In this case, T2 cannot see the data added by **COPY** of T1.

Scenario 2:

- **READ COMMITTED** level

  T1 is started but not committed. At this time, T2 is started. **COPY** of T1 is complete and T1 is committed. In this case, T2 can see the data added by **COPY** of T1.

- **REPEATABLE READ** level

  T1 is started but not committed. At this time, T2 is started. **COPY** of T1 is complete and T1 is committed. In this case, T2 cannot see the data added by **COPY** of T1.

9 Performance Tuning

9.1 Performance Tuning Process

To fine-tune GaussDB performance, you need to identify performance bottlenecks, adjust key parameters, and optimize SQL statements. During the tuning, factors such as system resources, throughput, and loads can be used to locate and analyze performance problems to ensure that the system performance is acceptable.

Multiple factors need to be considered during GaussDB performance optimization. Therefore, optimization personnel must have a wide and deep understanding of the system software architecture, software and hardware configurations, database configuration parameters, concurrency control (The current feature is a lab feature. Contact Huawei technical support before using it.), query processing, and database applications.

NOTICE

During performance optimization, the database needs to be restarted sometimes, which may interrupt current services. Therefore, after services are brought online, if the database needs to be restarted for performance optimization, submit an application to the management department for the operation window time. The operation can be performed only after the application is approved.

Tuning Process

Figure 9-1 shows the procedure of performance tuning.

Figure 9-1 GaussDB performance optimization process

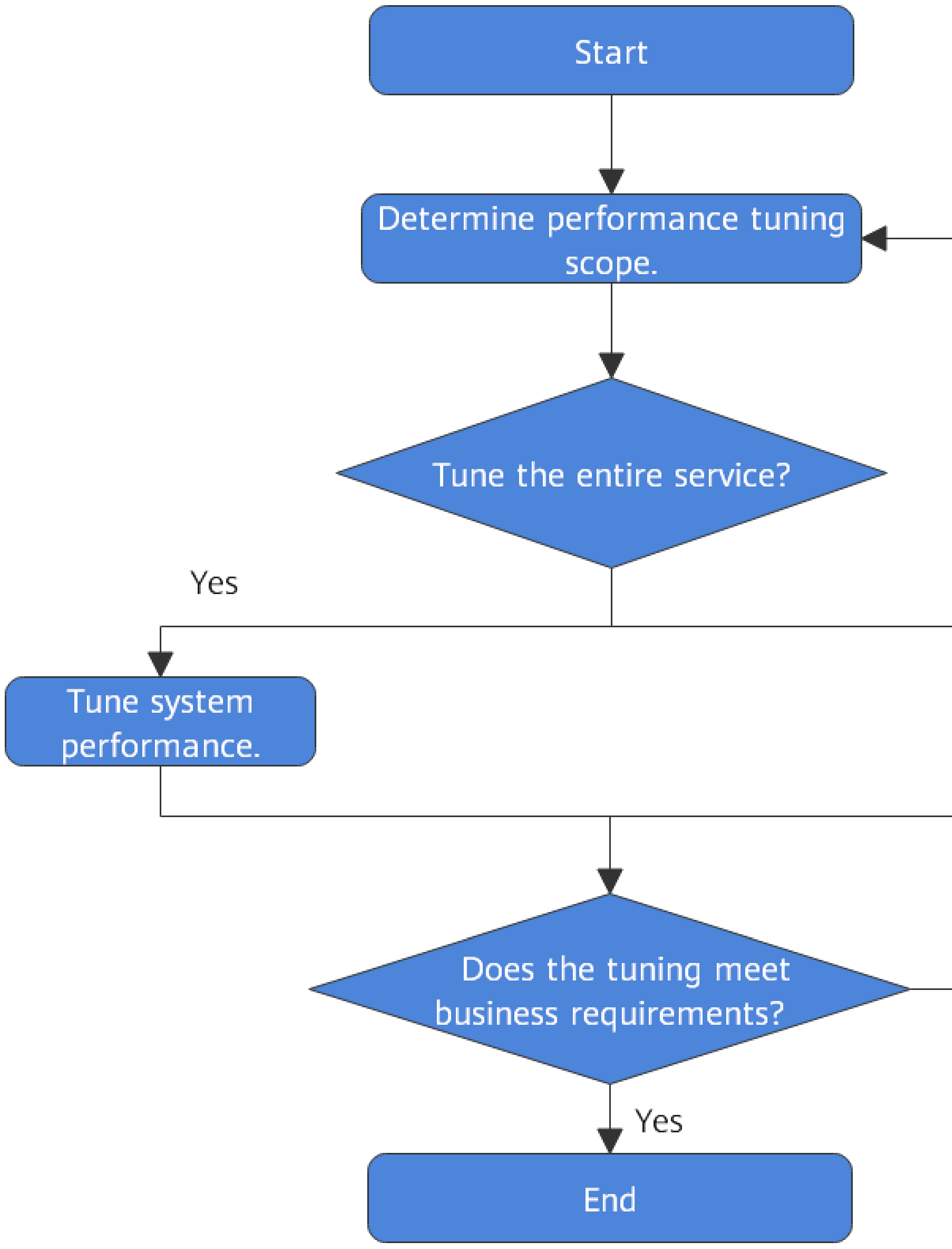

Start
Determine performance tuning scope.
Tune the entire service?
Yes
Tune system performance.
Does the tuning meet business requirements?
Yes
End

Table 9-1 lists the details about each phase.

Table 9-1 GaussDB performance optimization process

Phase	Description
Determining the Scope of Performance Tuning	The CPU, memory, I/O, and network resource usage of each node in the database are obtained to check whether these resources are fully used and whether any bottleneck exists.
SQL Optimization	The SQL statements are analyzed to determine whether any optimization can be performed. Analysis of SQL statements comprises: ● Generating table statistics using **ANALYZE**: The **ANALYZE** statement collects statistics about the database table content. Statistical results are stored in the system catalog **PG_STATISTIC**. The execution plan generator uses these statistics to determine which one is the most effective execution plan. ● Analyzing the execution plan: The **EXPLAIN** statement displays the execution plan of SQL statements, and the **EXPLAIN PERFORMANCE** statement displays the execution time of each operator in SQL statements. ● Identifying the root causes of issues: Identifies possible causes by analyzing the execution plan and performs specific optimization by modifying database-level SQL optimization parameters. ● Compiling better SQL statements: Compiles better SQL statements in the scenarios, such as cache of intermediate and temporary data for complex queries, result set cache, and result set combination.

9.2 Determining the Scope of Performance Tuning

Database performance tuning often happens when users are not satisfied with the service execution efficiency and want to improve the efficiency. The database performance is affected by many factors as described in section **Performance Elements**. Therefore, performance tuning is a complex process and sometimes cannot be systematically described or explained. It depends more on the database administrator's experience. However, this section still attempts to illustrate the performance tuning methods that can be referred to by application development personnel and new GaussDB administrators.

Performance Elements

There are multiple performance factors that affect the database performance. Knowing these factors can help you identify and analyze performance-associated issues.

- System resources

  Database performance greatly relies on disk I/O and memory usage. To accurately set performance counters, you need to have a knowledge of the basic performance of the hardware deployed in database. Performance of hardware, such as the CPU, hard disk, disk controller, memory, and network interfaces, greatly affects database running speed.

- Load

  The load indicates the total database system demands and it changes over time. The overall load contains user queries, applications, concurrent jobs, transactions, and system commands transferred at any time. For example, the system load increases if multiple users are executing multiple queries. The load will significantly affect the database performance. Identifying load peak hours helps improve resource utilization so that tasks are executed effectively.

- Throughput

  The data processing capability of a database is defined by its throughput. Database throughput is measured by the number of queries or processed transactions per second or by the average response time. The database processing capacity is closely related to the underlying system performance (disk I/O, CPU speed, and storage bandwidth). You need to know about the hardware performance before setting a target throughput.

- Competition

  Competition indicates that two or more load components try to use system resources in a conflicting way. For example, competition occurs when multiple queries attempt to update the same data at the same time, or when a large number of loads compete for system resources. When competition increases, the throughput decreases.

- Optimization

  The database optimization can affect the performance of the whole system. Before executing the SQL statements, configuring database parameters, designing tables, and performing data distribution, enable the database query optimizer can help you obtain the most efficient execution plan.

Determining the Tuning Scope

Performance tuning depends on the usage of hardware resources, such as the CPU, memory, I/O, and network of each node in database. Check whether these resources are fully utilized, and whether any bottlenecks exist, and then perform performance tuning as required.

- If a resource reaches the bottleneck:
  a. Check whether the key OS parameters and database parameters are properly set.
  b. Find the resource consuming SQL statements by querying the most time-consuming SQL statements and unresponsive SQL statements, and then perform **SQL Optimization**.
- If no resource reaches the bottleneck, the system performance can be improved. In this case, query the most time-consuming SQL statements and the unresponsive SQL statements, and then perform **SQL Optimization** as required.

9.2.1 Querying SQL Statements That Affect Performance Most

This section describes how to query SQL statements whose execution takes a long time, leading to poor system performance.

Procedure

Step 1 Connect to a database. For details, see **Connecting to a Database**.

Step 2 Query the statements that are run for a long time in the database.

```
SELECT current_timestamp - query_start AS runtime, datname, usename, query FROM pg_stat_activity
where state != 'idle' ORDER BY 1 desc;
```

The command output lists the query statements in descending order by their execution duration length. The first record is the query statement that takes the longest time for execution. The returned result contains SQL statements invoked by the system and SQL statements run by users. Find the statements that were run by users and took a long time.

Alternatively, you can set **current_timestamp - query_start** to be greater than a threshold to identify query statements that are executed for a duration longer than this threshold.

```
SELECT query FROM pg_stat_activity WHERE current_timestamp - query_start > interval '1 days';
```

Step 3 Set the parameter **track_activities** to **on**.

```
SET track_activities = on;
```

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

Step 4 View the running query statements.

The **pg_stat_activity** view is used as an example here.

```
SELECT datname, usename, state FROM pg_stat_activity;
 datname  | usename | state  |
----------+---------+--------+
 postgres |   omm   | idle   |
 postgres |   omm   | active |
(2 rows)
```

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

To identify only active query statements, run the following command:

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

Step 5 Analyze the status of the query statements that were run for a long time.

- If the query statement is normal, wait until the execution of the query statement is complete.
- If a query statement is blocked, run the following command to view this query statement:

  ```
  SELECT datname, usename, state, query FROM pg_stat_activity WHERE waiting = true;
  ```

  The query statement is displayed. It is requesting a lock resource that may be held by another session, and is waiting for the lock resource to be released by the session.

  NOTE

  Only when the query is blocked by internal lock resources, the **waiting** column is **true**. In most cases, blocks happen when query statements are waiting for lock resources to be released. However, query statements may be blocked due to write and timers operations. Such blocked queries are not displayed in the **pg_stat_activity** view.

----End

9.2.2 Checking Blocked Statements

During database running, query statements are blocked in some service scenarios and run for an excessively long time. In this case, you can forcibly terminate the faulty session.

Procedure

Step 1 Connect to a database. For details, see **Connecting to a Database**.

Step 2 View blocked query statements and details about the tables and schemas that block the query statements.

```
SELECT w.query as waiting_query,
w.pid as w_pid,
w.usename as w_user,
l.query as locking_query,
l.pid as l_pid,
l.usename as l_user,
t.schemaname || '.' || t.relname as tablename
from pg_stat_activity w join pg_locks l1 on w.pid = l1.pid
and not l1.granted join pg_locks l2 on l1.relation = l2.relation
and l2.granted join pg_stat_activity l on l2.pid = l.pid join pg_stat_user_tables t on l1.relation = t.relid
where w.waiting;
```

The thread ID, user details, query status, as well as details about the tables and schemas that block the query statements are returned.

Step 3 Run the following command to terminate the required session, where **139834762094352** is the thread ID:

```
SELECT PG_TERMINATE_BACKEND(139834762094352);
```

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

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

If information similar to the following is displayed, a user is attempting to terminate the session, and the session will be reconnected rather than being terminated.

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

NOTE

If the **PG_TERMINATE_BACKEND** function is used to terminate the background threads of the session, the **gsql** client will be reconnected rather than be logged out.

----End

9.3 SQL Optimization

The aim of SQL optimization is to maximize the utilization of resources, including CPU, memory, and disk I/O. All optimization methods are intended for resource utilization. To maximize resource utilization is to run SQL statements as efficiently as possible to achieve the highest performance at a lower cost. For example, when performing a typical point query, you can use a Seq Scan and a filter (that is, read every tuple and point query conditions for match). You can also use an Index Scan, which can be implemented at a lower cost but achieve the same effect.

You can determine a proper database deployment solution and table definition based on hardware resources and service characteristics. This is the basis of meeting performance requirements. The following performance tuning sections assume that you have finished installation based on a proper database solution in the software installation guide and performed database design based on the guide for database design and development.

9.3.1 Query Execution Process

The process from receiving SQL statements to the statement execution by the SQL engine is shown in **Figure 9-2** and described in **Table 9-2**. The texts in red are steps where database administrators can optimize queries.

Figure 9-2 Execution process of query-related SQL statements by the SQL engine

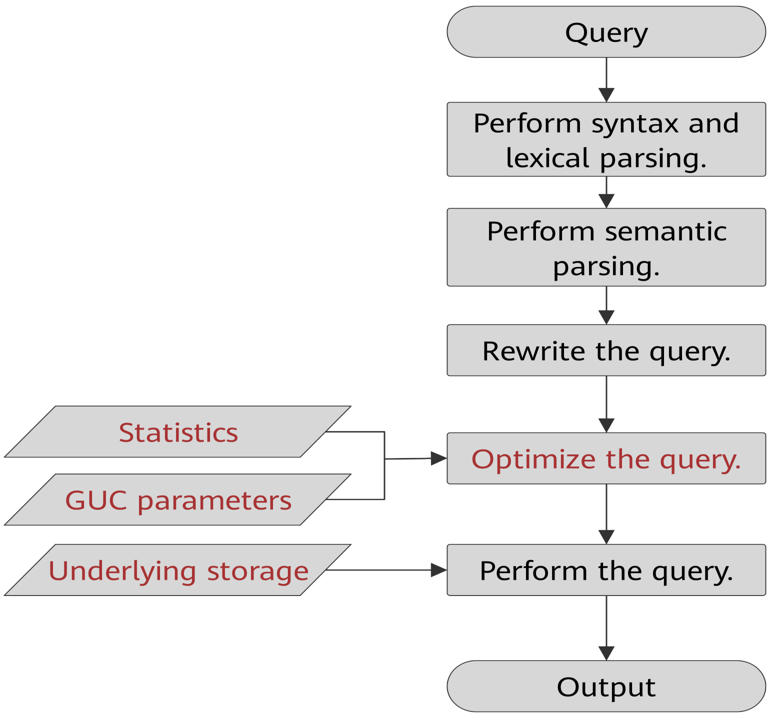


Table 9-2 Execution process of query-related SQL statements by the SQL engine

Step	Description
1. Perform syntax and lexical parsing.	Converts the input SQL statements from the string data type to the formatted structure stmt based on the specified SQL statement rules.
2. Perform semantic parsing.	Converts the formatted structure obtained from the previous step into objects that can be recognized by the database.

Step	Description
3. Rewrite the query statements.	Converts the output of the previous step into the structure that optimizes the query execution.
4. Optimize the query.	Determines the execution mode of SQL statements (the execution plan) based on the result obtained from the previous step and the internal database statistics. For details about how the internal database statistics and GUC parameters affect the query optimization (execution plan), see **Optimizing Queries Using Statistics** and **Optimizing Queries Using GUC parameters**.
5. Perform the query.	Executes the SQL statements based on the execution path specified in the previous step. Selecting a proper underlying storage mode improves the query execution efficiency. For details, see **Optimizing Queries Using the Underlying Storage**.

Optimizing Queries Using Statistics

The GaussDB optimizer is a typical Cost-based Optimization (CBO). By using CBO, the database calculates the number of tuples and the execution cost for each step under each execution plan based on the number of table tuples, column width, null record ratio, and characteristic values, such as distinct, MCV, and HB values, and certain cost calculation methods. The database then selects the execution plan that takes the lowest cost for the overall execution or for the return of the first tuple. These characteristic values are the statistics, which is the core for optimizing a query. Accurate statistics helps the planner select the most appropriate query plan. Generally, you can collect statistics of a table or that of some columns in a table using **ANALYZE**. You are advised to periodically execute **ANALYZE** or execute it immediately after you modified most contents in a table.

Optimizing Queries Using GUC parameters

Optimizing queries aims to select an efficient execution mode.

Take the following SQL statement as an example:

```
select count(1)
from customer inner join store_sales on (ss_customer_sk = c_customer_sk);
```

During execution of **customer inner join store_sales**, GaussDB supports nested loop, merge join, and hash join. The optimizer estimates the result set sizes and the execution cost for each join mode based on the statistics on the **customer** and **store_sales** tables. It then compares the costs and selects the one costing the least.

As described in the preceding content, the execution cost is calculated based on certain methods and statistics. If the actual execution cost cannot be accurately estimated, you need to optimize the execution plan by setting the GUC parameters.

Optimizing Queries Using the Underlying Storage

GaussDB supports row- and column-store tables. The selection of an underlying storage mode strongly depends on specific customer service scenarios. You are advised to use column-store tables for computing service scenarios (mainly involving association and aggregation operations) and row-store tables for service scenarios, such as point queries and massive **UPDATE** or **DELETE** executions.

Optimization methods of each storage mode will be described in detail below.

Optimizing Queries by Rewriting SQL Statements

Besides the preceding methods that improve the performance of the execution plan generated by the SQL engine, database administrators can also enhance SQL statement performance by rewriting SQL statements while retaining the original service logic based on the execution mechanism of the database and abundant practices.

This requires that database administrators know the customer services well and have professional knowledge of SQL statements. Below chapters will describe some common SQL rewriting scenarios.

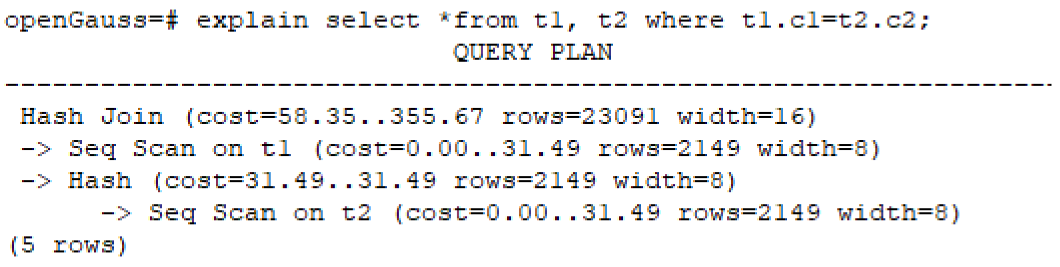
9.3.2 Introduction to the SQL Execution Plan

9.3.2.1 Overview

The SQL execution plan is a node tree, which displays detailed procedure when GaussDB runs an SQL statement. A database operator indicates one step.

You can run the **EXPLAIN** command to view the execution plan generated for each query by an optimizer. The output of **EXPLAIN** has one row for each execution node, showing the basic node type and the cost estimation that the optimizer made for the execution of this node, as shown in **Figure 9-3**.

Figure 9-3 SQL execution plan example



```
openGauss=# explain select *from t1, t2 where t1.c1=t2.c2;
                            QUERY PLAN
-----------------------------------------------------------------
 Hash Join (cost=58.35..355.67 rows=23091 width=16)
 -> Seq Scan on t1 (cost=0.00..31.49 rows=2149 width=8)
 -> Hash (cost=31.49..31.49 rows=2149 width=8)
      -> Seq Scan on t2 (cost=0.00..31.49 rows=2149 width=8)
(5 rows)
```

- Nodes at the bottom level are scan nodes. They scan tables and return raw rows. The types of scan nodes (sequential scans and index scans) vary depending on the table access methods. Objects scanned by the bottom layer nodes may not be row-store data (not directly read from a table), such as **VALUES** clauses and functions that return rows, which have their own types of scan nodes.
- If the query requires join, aggregation, sorting, or other operations on the raw rows, there will be other nodes above the scan nodes to perform these

operations. In addition, there is more than one way to perform these operations, so different types of execution nodes may be displayed here.

- The first row (the upper-layer node) estimates the total execution cost of the execution plan. Such an estimate indicates the value that the optimizer tries to minimize.

Execution Plan Information

In addition to setting different display formats for an execution plan, you can use different **EXPLAIN** syntax to display execution plan information in detail. The following lists the common **EXPLAIN** syntax. For details about more **EXPLAIN** syntax, see **EXPLAIN**.

- EXPLAIN *statement*: only generates an execution plan and does not execute. The *statement* indicates SQL statements.
- EXPLAIN ANALYZE *statement*: generates and executes an execution plan, and displays the execution summary. Then actual execution time statistics are added to the display, including the total elapsed time expended within each plan node (in milliseconds) and the total number of rows it actually returned.
- EXPLAIN PERFORMANCE *statement*: generates and executes the execution plan, and displays all execution information.

To measure the run time cost of each node in the execution plan, the current execution of **EXPLAIN ANALYZE** or **EXPLAIN PERFORMANCE** adds profiling overhead to query execution. Running **EXPLAIN ANALYZE** or **EXPLAIN PERFORMANCE** on a query sometimes takes longer time than executing the query normally. The amount of overhead depends on the nature of the query, as well as the platform being used.

Therefore, if an SQL statement is not finished after being running for a long time, run the **EXPLAIN** statement to view the execution plan and then locate the fault. If the SQL statement has been properly executed, run the **EXPLAIN ANALYZE** or **EXPLAIN PERFORMANCE** statement to check the execution plan and information to locate the fault.

9.3.2.2 Description

As described in **Overview**, **EXPLAIN** displays the execution plan, but will not actually run SQL statements. **EXPLAIN ANALYZE** and **EXPLAIN PERFORMANCE** both will actually run SQL statements and return the execution information. This section describes the execution plan and execution information in detail.

Execution Plans

The following SQL statement is used as an example:

```
SELECT * FROM t1, t2 WHERE t1.c1 = t2.c2;
```

Run the **EXPLAIN** command and the output is as follows:

```
openGauss=# explain select *from t1, t2 where t1.c1=t2.c2;
                              QUERY PLAN
-------------------------------------------------------------------
 Hash Join (cost=58.35..355.67 rows=23091 width=16)
 -> Seq Scan on t1 (cost=0.00..31.49 rows=2149 width=8)
 -> Hash (cost=31.49..31.49 rows=2149 width=8)
      -> Seq Scan on t2 (cost=0.00..31.49 rows=2149 width=8)
(5 rows)
```

Interpretation of the execution plan level (vertical):

1. Layer 1: **Seq Scan on t2**

   The table scan operator scans the table **t2** using **Seq Scan**. At this layer, data in the table **t2** is read from a buffer or disk, and then transferred to the upper-layer node for calculation.

2. Layer 2: **Hash**

   Hash operator. It is used to calculate the hash value of the operator transferred from the lower layer for subsequent hash join operations.

3. Layer 3: **Seq Scan on t1**

   The table scan operator scans the table **t1** using **Seq Scan**. At this layer, data in the table **t1** is read from a buffer or disk, and then transferred to the upper-layer node for hash join calculation.

4. Layer 4: **Hash Join**

   Join operator. It is used to join data in the **t1** and **t2** tables using the hash join method and output the result data.

Keywords in the execution plan:

1. Table access modes
   - Seq Scan

     Scans all rows of the table in sequence.

   - Index Scan

     The optimizer uses a two-step plan: the child plan node visits an index to find the locations of rows matching the index condition, and then the upper plan node actually fetches those rows from the table itself. Fetching rows separately is much more expensive than reading them sequentially, but because not all pages of the table have to be visited, this is still cheaper than a sequential scan. The upper-layer planning node sorts index-identified rows based on their physical locations before reading them. This minimizes the independent capturing overhead.

     If there are separate indexes on multiple columns referenced in **WHERE**, the optimizer might choose to use an **AND** or **OR** combination of the indexes. However, this requires the visiting of both indexes, so it is not necessarily a win compared to using just one index and treating the other condition as a filter.

     The following index scans featured with different sorting mechanisms are involved:

     - Bitmap index scan

       Fetches data pages using a bitmap.

- Index scan using index_name

  Fetches table rows in index order, which makes them even more expensive to read. However, there are so few rows that the extra cost of sorting the row locations is unnecessary. This plan type is used mainly for queries fetching just a single row and queries having an **ORDER BY** condition that matches the index order, because no extra sorting step is needed to satisfy **ORDER BY**.

2. Table connection modes
   - Nested Loop

     A nested loop is used for queries that have a smaller data set connected. In a nested loop join, the foreign table drives the internal table and each row returned from the foreign table should have a matching row in the internal table. The returned result set of all queries should be less than 10,000. The table that returns a smaller subset will work as a foreign table, and indexes are recommended for connection columns of the internal table.
   - (Sonic) Hash Join

     A hash join is used for large tables. The optimizer uses a hash join, in which rows of one table are entered into an in-memory hash table, after which the other table is scanned and the hash table is probed for matches to each row. Sonic and non-Sonic hash joins differ in their hash table structures, which do not affect the execution result set.
   - Merge Join

     In most cases, the execution performance of a merge join is lower than that of a hash join. However, if the source data has been pre-sorted and no more sorting is needed during the merge join, its performance excels.
3. Operators
   - sort

     Sorts the result set.
   - filter

     The **EXPLAIN** output shows the **WHERE** clause being applied as a **Filter** condition attached to the **Seq Scan** plan node. This means that the plan node checks the condition for each row it scans, and returns only the ones that meet the condition. The estimated number of output rows has been reduced because of the **WHERE** clause. However, the scan will still have to visit all 10,000 rows, as a result, the cost is not decreased. It increases a bit (by 10,000 x **cpu_operator_cost**) to reflect the extra CPU time spent on checking the **WHERE** condition.
   - LIMIT

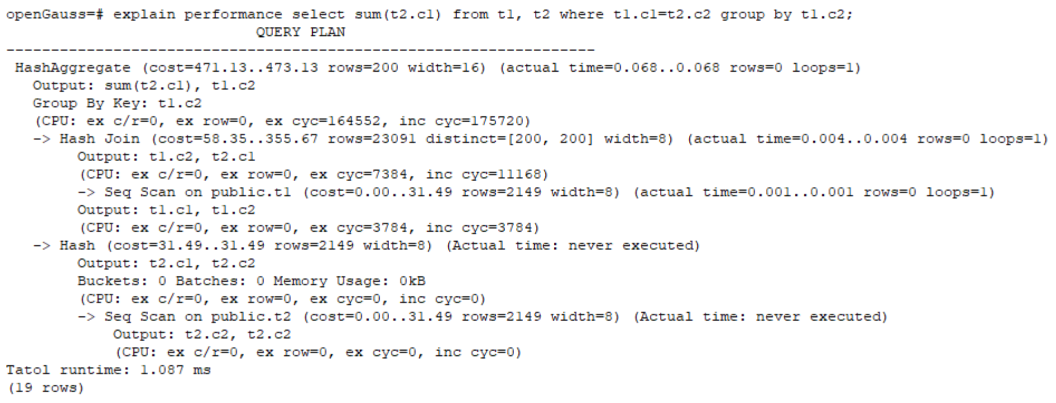
     Limits the number of output execution results. If a **LIMIT** condition is added, not all rows are retrieved.

Execution Information

The following SQL statement is used as an example:

```
select sum(t2.c1) from t1,t2 where t1.c1=t2.c2 group by t1.c2;
```

The output of running **EXPLAIN PERFORMANCE** is as follows:



```
openGauss=# explain performance select sum(t2.c1) from t1, t2 where t1.c1=t2.c2 group by t1.c2;
                              QUERY PLAN
------------------------------------------------------------------
 HashAggregate (cost=471.13..473.13 rows=200 width=16) (actual time=0.068..0.068 rows=0 loops=1)
   Output: sum(t2.c1), t1.c2
   Group By Key: t1.c2
   (CPU: ex c/r=0, ex row=0, ex cyc=164552, inc cyc=175720)
   -> Hash Join (cost=58.35..355.67 rows=23091 distinct=[200, 200] width=8) (actual time=0.004..0.004 rows=0 loops=1)
        Output: t1.c2, t2.c1
        (CPU: ex c/r=0, ex row=0, ex cyc=7384, inc cyc=11168)
        -> Seq Scan on public.t1 (cost=0.00..31.49 rows=2149 width=8) (actual time=0.001..0.001 rows=0 loops=1)
        Output: t1.c1, t1.c2
        (CPU: ex c/r=0, ex row=0, ex cyc=3784, inc cyc=3784)
   -> Hash (cost=31.49..31.49 rows=2149 width=8) (Actual time: never executed)
        Output: t2.c1, t2.c2
        Buckets: 0 Batches: 0 Memory Usage: 0kB
        (CPU: ex c/r=0, ex row=0, ex cyc=0, inc cyc=0)
        -> Seq Scan on public.t2 (cost=0.00..31.49 rows=2149 width=8) (Actual time: never executed)
            Output: t2.c2, t2.c2
            (CPU: ex c/r=0, ex row=0, ex cyc=0, inc cyc=0)
Tatol runtime: 1.087 ms
(19 rows)
```

9.3.3 Tuning Process

You can analyze slow SQL statements to optimize them.

Procedure

Step 1 Collect all table statistics associated with the SQL statements. In a database, statistics indicate the source data of a plan generated by a planner. If no statistics are available or out of date, the execution plan may seriously deteriorate, leading to low performance. According to past experience, about 10% performance problems occurred because no statistics are collected. For details, see **Updating Statistics**.

Step 2 View the execution plan to find out the cause. If the SQL statements have been running for a long period of time and not ended, run the **EXPLAIN** statement to view the execution plan and then locate the fault. If the SQL statement has been properly executed, run the **EXPLAIN ANALYZE** or **EXPLAIN PERFORMANCE** statement to check the execution plan and information to locate the fault. For details about the execution plan, see **Introduction to the SQL Execution Plan**.

Step 3 Review and modify a table definition. For details, see **Reviewing and Modifying a Table Definition**.

Step 4 For details about **EXPLAIN** or **EXPLAIN PERFORMANCE**, the reason why SQL statements are slowly located, and how to solve this problem, see **Typical SQL Optimization Methods**.

Step 5 Generally, some SQL statements can be converted to its equivalent statements in all or certain scenarios by rewriting queries. SQL statements are simpler after they are rewritten. Some execution steps can be simplified to improve the performance. Query rewriting methods are universal in all databases. **Experience in Rewriting SQL Statements** describes several tuning methods by rewriting SQL statements.

----End

9.3.4 Updating Statistics

In a database, statistics indicate the source data of a plan generated by a planner. If no statistics are available or out of date, the execution plan may seriously deteriorate, leading to low performance.

Background

The **ANALYZE** statement collects statistic about table contents in databases, which will be stored in the **PG_STATISTIC** system catalog. Then, the query optimizer uses the statistics to work out the most efficient execution plan.

After executing batch insertions and deletions, you are advised to run the **ANALYZE** statement on the table or the entire library to update statistics. By default, 30,000 rows of statistics are sampled. That is, the default value of the GUC parameter **default_statistics_target** is **100**. If the total number of rows in the table exceeds 1,600,000, you are advised to set **default_statistics_target** to **-2**, indicating that 2% of the statistics are collected.

For an intermediate table generated during the execution of a batch script or stored procedure, you also need to run the **ANALYZE** statement.

If there are multiple inter-related columns in a table and the conditions or grouping operations based on these columns are involved in the query, collect statistics about these columns so that the query optimizer can accurately estimate the number of rows and generate an effective execution plan. (The current feature is a lab feature. Contact Huawei technical support before using it.)

Procedure

Run the following commands to update the statistics about a table or the entire database:

```
ANALYZE tablename;                    --Update statistics about a table.
ANALYZE;                          ---Update statistics about the entire database.
```

Run the following statements to perform statistics-related operations on multiple columns (The current feature is a lab feature. Contact Huawei technical support before using it.):

```
ANALYZE tablename ((column_1, column_2));                    --Collect statistics about column_1 and
column_2 of tablename.

ALTER TABLE tablename ADD STATISTICS ((column_1, column_2));    --Declare statistics about column_1
and column_2 of tablename.
ANALYZE tablename;                                          --Collect statistics about one or more columns.

ALTER TABLE tablename DELETE STATISTICS ((column_1, column_2)); --Delete statistics about column_1
and column_2 of tablename or their statistics declaration.
```

NOTICE

After the statistics are declared for multiple columns by running the **ALTER TABLE** *tablename* **ADD STATISTICS** statement, the system collects the statistics about these columns next time **ANALYZE** is performed on the table or the entire database.

To collect the statistics, run the **ANALYZE** statement.

NOTE

Use **EXPLAIN** to show the execution plan of each SQL statement. If **rows=10** (the default value, probably indicating that the table has not been analyzed) is displayed in the **SEQ SCAN** output of a table, run the **ANALYZE** statement for this table.

9.3.5 Reviewing and Modifying a Table Definition

9.3.5.1 Overview

To properly define a table, you must:

1. **Reduce the data volume scanned** by using the partition pruning mechanism.
2. **Minimize random I/Os** by using clustering or partial clustering.

The table definition is created during the database design and is reviewed and modified during the SQL statement optimization.

9.3.5.2 Selecting a Storage Model

During database design, some key factors about table design will greatly affect the subsequent query performance of the database. Table design affects data storage as well. Scientific table design reduces I/O operations and minimizes memory usage, improving the query performance.

Selecting a model for table storage is the first step of table definition. Select a proper storage model for your service based on the following table.

Storage Model	Application Scenario
Row storage	Point queries (simple index-based queries that only return a few records) Scenarios requiring frequent addition, deletion, and modification operations
Column storage	Statistics analysis query, in which operations, such as group and join, are performed many times

9.3.5.3 Using PCKs

The PCK is the column-store-based technology. It can minimize or maximize sparse indexes to quickly filter base tables. You are advised to select a maximum of two columns as PCKs. Use the following principles to specify PCKs:

1. The selected PCKs must be restricted by simple expressions in base tables. Such constraints are usually represented by *col op const*, in which *col* indicates the column name, *op* indicates operators, (including =, >, >=, <=, and <), and *const* indicates constants.
2. Select columns that are frequently selected (to filter much more undesired data) in simple expressions.
3. List the less frequently selected columns on the top.
4. List the columns of the enumerated type at the top.

9.3.5.4 Using Partitioned Tables

Partitioning refers to splitting what is logically one large table into smaller physical pieces based on specific schemes. The table based on the logic is called a partitioned table, and a physical piece is called a partition. Data is stored in physical partitions not the logical table. A partitioned table has the following advantages over an ordinary table:

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

Partitioned tables supported by GaussDB are level-1 and level-2 partitioned tables. Level-1 partitioned tables include range partitioned tables, interval partitioned tables, list partitioned tables, and hash partitioned tables. Level-2 partitioned tables include nine combinations of any two of range partitioned tables, list partitioned tables, and hash partitioned tables.

- Range partitioned table: Data in different ranges is mapped to different partitions. The range is determined by the partition key specified during the partitioned table creation. The partition key is usually a date. For example, sales data is partitioned by month.
- Interval partitioned table: a special type of range partitioned tables. Compared with range partitioned tables, interval value definition is added. When no matching partition can be found for an inserted record, a partition can be automatically created based on the interval value.
- List partitioned table: Key values contained in the data are stored in different partitions, and the data is mapped to each partition in sequence. The key values contained in the partitions are specified when the partitioned table is created.
- Hash partitioned table: Data is mapped to each partition based on the internal hash algorithm. The number of partitions is specified when the partitioned table is created.
- Level-2 partitioned table: a partitioned table obtained by randomly combining range partitioning, list partitioning, and hash partitioning. Both level-1 and level-2 partitions can be defined in the preceding three ways.

9.3.5.5 Selecting a Data Type

Use the following principles to select efficient data types:

1. **Select data types that facilitate data calculation.**

   Generally, the calculation of integers (including common comparison calculations, such as =, >, <, ≥, ≤, and ≠ and group by) is more efficient than that of strings and floating point numbers. For example, if you need to perform a point query on a column-store table whose numeric column is used as a filter condition, the query will take over 10s. If you change the data type from **NUMERIC** to **INT**, the query duration will be reduced to 1.8s.

2. **Select data types with a short length.**

   Data types with short length reduce both the data file size and the memory used for computing, improving the I/O and computing performance. For example, use **SMALLINT** instead of **INT**, and **INT** instead of **BIGINT**.

3. **Use the same data type for a join.**

   You are advised to use the same data type for a join. To join columns with different data types, the database needs to convert them to the same type, which leads to additional performance overheads.

9.3.6 Typical SQL Optimization Methods

SQL optimization involves continuous analysis and trying. Queries are run before they are used for services to determine whether the performance meets requirements. If it does not, queries will be optimized by **checking the execution plan** and identifying the causes. Then, the queries will be run and optimized again until they meet the requirements.

9.3.6.1 Optimizing SQL Self-Diagnosis

Performance issues may occur when you query data or run the **INSERT**, **DELETE**, **UPDATE**, or **CREATE TABLE AS** statement. In this case, you can query the **warning** column in the **PG_CONTROL_GROUP_CONFIG** and **GS_SESSION_MEMORY_DETAIL** views to obtain reference for performance optimization.

Alarms that can trigger SQL self diagnosis depend on the settings of **resource_track_level**. If **resource_track_level** is set to **query**, alarms about the failures in collecting column statistics and pushing down SQL statements will trigger the diagnosis. If **resource_track_level** is set to **operator**, all alarms will trigger the diagnosis.

Whether a SQL plan will be diagnosed depends on the settings of **resource_track_cost**. A SQL plan will be diagnosed only if its execution cost is greater than **resource_track_cost**. You can use the **EXPLAIN** keyword to check the plan execution cost.

The SQL self-diagnosis function is affected by the **enable_analyze_check** parameter. Ensure that the function is enabled before using it.

If a large number of statements are executed, certain data may fail to be collected due to memory control. In this case, you can increase the value of **instr_unique_sql_count**.

Alarms

Currently, performance alarms will be reported when statistics about one or multiple columns are not collected.

An alarm will be reported if some column statistics are not collected. The current feature is a lab feature. Contact Huawei technical support before using it. For details about the optimization, see **Updating Statistics** and **Optimizing Statistics**.

Example alarms:

No statistics about a table are not collected.

```
Statistic Not Collect:
    schema_test.t1
```

The statistics about a single column are not collected.

```
Statistic Not Collect:
    schema_test.t2(c1,c2)
```

The statistics about multiple columns are not collected.

```
Statistic Not Collect:
    schema_test.t3((c1,c2))
```

The statistics about a single column and multiple columns are not collected.

```
Statistic Not Collect:
    schema_test.t4(c1,c2)    schema_test.t4((c1,c2))
```

Restrictions

1. An alarm contains a maximum of 2048 characters. If the length of an alarm exceeds this value (for example, a large number of long table names and column names are displayed in the alarm when their statistics are not collected), a warning instead of an alarm will be reported.
   ```
   WARNING, "Planner issue report is truncated, the rest of planner issues will be skipped"
   ```
2. If a query statement contains the **Limit** operator, alarms of operators lower than **Limit** will not be reported.

9.3.6.2 Optimizing Subqueries

Background

When an application runs a SQL statement to operate the database, a large number of subqueries are used because they are more clear than table join. Especially in complicated query statements, subqueries have more complete and independent semantics, which makes SQL statements clearer and easier to understand. Therefore, subqueries are widely used.

In GaussDB, subqueries can also be called sublinks based on the location of subqueries in SQL statements.

- Subquery: corresponds to a range table (RangeTblEntry) in the query parse tree. That is, a subquery is a **SELECT** statement following immediately after the **FROM** keyword.
- Sublink: corresponds to an expression in the query parsing tree. That is, a sublink is a statement in the **WHERE** or **ON** clause or in the target list.

  In conclusion, a subquery is a RangeTblEntry and a sublink is an expression in the query parsing tree. A sublink can be found in constraint conditions and expressions. In GaussDB, sublinks can be classified into the following types:
  - exist_sublink: corresponds to the **EXIST** and **NOT EXIST** statements.
  - any_sublink: corresponds to the *op* **ANY(SELECT...)** statement. *op* can be the **<**, **>**, or **=** operator. **IN/NOT IN (SELECT...)** also belongs to this type.
  - all_sublink: corresponds to the *op* **ALL(SELECT...)** statement. *op* can be the **<**, **>**, or **=** operator.
  - rowcompare_sublink: corresponds to the **RECORD** *op* **(SELECT...)** statement.
  - expr_sublink: corresponds to the **(SELECT** *with a single target list item...***)** statement.

- array_sublink: corresponds to the **ARRAY(SELECT...)** statement.
- cte_sublink: corresponds to the **WITH(...)** query statement.

The exist_sublink and any_sublink are pulled up by the optimization engine of GaussDB. In addition, expr_sublink can also be pulled up. However, because of the flexible use of subqueries in SQL statements, complex subqueries may affect query performance. If you do not want to pull up expr_sublink, set the GUC parameter **rewrite_rule**. For details, see **Other Optimizer Options**.
Subqueries are classified into non-correlated subqueries and correlated subqueries.

- **Non-correlated subqueries**

  The execution of a subquery is independent from attributes of the outer query. In this way, a subquery can be executed before outer queries.

  Example:

```
select t1.c1,t1.c2
from t1
where t1.c1 in (
    select c2
    from t2
    where t2.c2 IN (2,3,4)
);
                          QUERY PLAN
---------------------------------------------------------------
 Hash Join
   Hash Cond: (t1.c1 = t2.c2)
   ->  Seq Scan on t1
         Filter: (c1 = ANY ('{2,3,4}'::integer[]))
   ->  Hash
         ->  HashAggregate
               Group By Key: t2.c2
               ->  Seq Scan on t2
                     Filter: (c2 = ANY ('{2,3,4}'::integer[]))
(9 rows)
```

- **Correlated subqueries**

  The execution of a subquery depends on some attributes (used as **AND** conditions of the subquery) of outer queries. In the following example, **t1.c1** in the **t2.c1 = t1.c1** condition is a correlated attribute. Such a subquery depends on outer queries and needs to be executed once for each outer query.

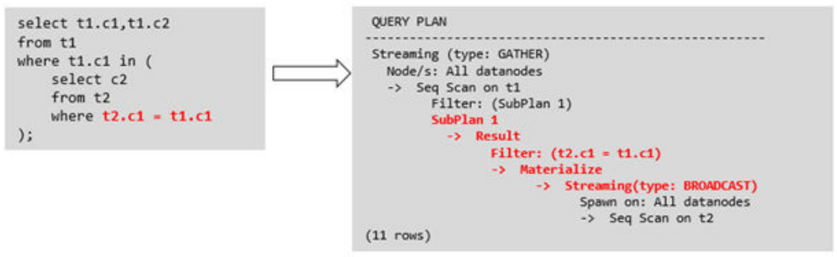
  Example:

```
select t1.c1,t1.c2
from t1
where t1.c1 in (
    select c2
    from t2
    where t2.c1 = t1.c1 AND t2.c2 in (2,3,4)
);
                           QUERY PLAN
-----------------------------------------------------------------------
 Seq Scan on t1
   Filter: (SubPlan 1)
   SubPlan 1
     ->  Seq Scan on t2
           Filter: ((c1 = t1.c1) AND (c2 = ANY ('{2,3,4}'::integer[])))
(5 rows)
```

Sublink Optimization on GaussDB

To optimize a sublink, a subquery is pulled up to join with tables in outer queries, preventing the subquery from being converted into a plan involving subplans and broadcast. You can run the **EXPLAIN** statement to check whether a sublink is converted into such a plan.

Example:



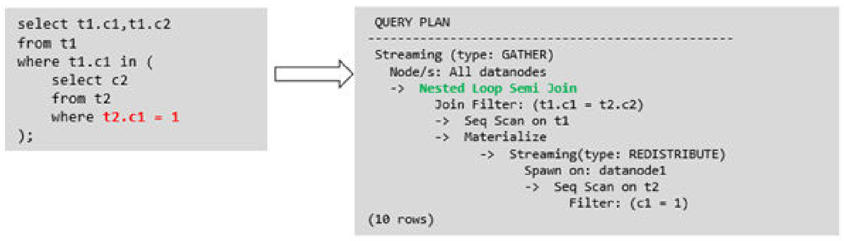
```
select t1.c1,t1.c2
from t1
where t1.c1 in (
    select c2
    from t2
    where t2.c1 = t1.c1
);
```

```
QUERY PLAN
------------------------------------------------------
Streaming (type: GATHER)
  Node/s: All datanodes
  ->  Seq Scan on t1
        Filter: (SubPlan 1)
        SubPlan 1
          ->  Result
                Filter: (t2.c1 = t1.c1)
                ->  Materialize
                      ->  Streaming(type: BROADCAST)
                            Spawn on: All datanodes
                            ->  Seq Scan on t2
(11 rows)
```

Replace the execution plan on the right of the arrow with the following execution plan:

```
QUERY PLAN
--------------------------------
Seq Scan on t1
Filter: (SubPlan 1)
SubPlan 1
->  Seq Scan on t2
Filter: (c1 = t1.c1)
(5 rows)
```

- **Sublink-release scenarios supported by GaussDB**
  - Pulling up the **IN** sublink
    - The subquery cannot contain columns in the outer query (columns in more outer queries are allowed).
    - The subquery cannot contain volatile functions.



```
select t1.c1,t1.c2
from t1
where t1.c1 in (
    select c2
    from t2
    where t2.c1 = 1
);
```

```
QUERY PLAN
------------------------------------------------------
Streaming (type: GATHER)
  Node/s: All datanodes
  ->  Nested Loop Semi Join
        Join Filter: (t1.c1 = t2.c2)
        ->  Seq Scan on t1
        ->  Materialize
              ->  Streaming(type: REDISTRIBUTE)
                    Spawn on: datanode1
                    ->  Seq Scan on t2
                          Filter: (c1 = 1)
(10 rows)
```

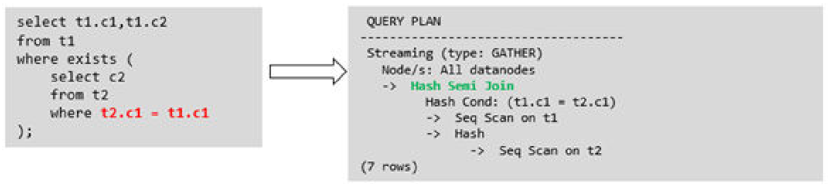
Replace the execution plan on the right of the arrow with the following execution plan:

```
QUERY PLAN
--------------------------------------
Hash Join
Hash Cond: (t1.c1 = t2.c2)
->  Seq Scan on t1
->  Hash
->  HashAggregate
Group By Key: t2.c2
->  Seq Scan on t2
Filter: (c1 = 1)
(8 rows)
```

  - Pulling up the **EXISTS** sublink

    The **WHERE** clause must contain a column in the outer query. Other parts of the subquery cannot contain the column. Other restrictions are as follows:

- The subquery must contain the **FROM** clause.
- The subquery cannot contain the **WITH** clause.
- The subquery cannot contain aggregate functions.
- The subquery cannot contain a **SET**, **SORT**, **LIMIT**, **WindowAgg**, or **HAVING** operation.
- The subquery cannot contain volatile functions.



```
select t1.c1,t1.c2
from t1
where exists (
     select c2
     from t2
     where t2.c1 = t1.c1
);
```

```
QUERY PLAN
-----------------------------------
Streaming (type: GATHER)
  Node/s: All datanodes
  -> Hash Semi Join
        Hash Cond: (t1.c1 = t2.c1)
        -> Seq Scan on t1
        -> Hash
              -> Seq Scan on t2
(7 rows)
```

Replace the execution plan on the right of the arrow with the following execution plan:

```
QUERY PLAN
-----------------------------------
Hash Join
Hash Cond: (t1.c1 = t2.c1)
-> Seq Scan on t1
-> Hash
-> HashAggregate
Group By Key: t2.c1
-> Seq Scan on t2
(7 rows)
```

– Pulling up an equivalent correlated query containing aggregate functions

The **WHERE** condition of the subquery must contain a column from the outer query. Equivalence comparison must be performed between this column and related columns in tables of the subquery. These conditions must be connected using **AND**. Other parts of the subquery cannot contain the column. Other restrictions are as follows:

- The columns in the expression in the **WHERE** condition of the subquery must exist in tables.
- After the **SELECT** keyword of the subquery, there must be only one output column. The output column must be an aggregate function (for example, **MAX**), and the parameter (for example, **t2.c2**) of the aggregate function cannot be columns of a table (for example, **t1**) in outer queries. The aggregate function cannot be **COUNT**.

  For example, the following subquery can be pulled up:

  ```
  select * from t1 where c1 >(
        select max(t2.c1) from t2 where t2.c1=t1.c1
  );
  ```

  The following subquery cannot be pulled up because the subquery has no aggregate function:

  ```
  select * from t1 where c1 >(
        select  t2.c1 from t2 where t2.c1=t1.c1
  );
  ```

  The following subquery cannot be pulled up because the subquery has two output columns:

```
select * from t1 where (c1,c2) >(
      select  max(t2.c1),min(t2.c2) from t2 where t2.c1=t1.c1
);
```

- The subquery must be a **FROM** clause.
- The subquery cannot contain a **GROUP BY**, **HAVING**, or **SET** operation.
- The subquery can only be an inner join.

  For example, the following subquery cannot be pulled up:

  ```
  select * from t1 where c1 >(
        select max(t2.c1) from t2 full join t3 on (t2.c2=t3.c2) where t2.c1=t1.c1
  );
  ```
- The target list of the subquery cannot contain the function that returns a set.
- The **WHERE** condition of the subquery must contain a column from the outer query. Equivalence comparison must be performed between this column and related columns in tables of the subquery. These conditions must be connected using **AND**. Other parts of the subquery cannot contain the column. For example, the following subquery can be pulled up:

  ```
  select * from t3 where t3.c1=(
         select t1.c1
         from t1 where c1 >(
                select max(t2.c1) from t2 where t2.c1=t1.c1
  ));
  ```

  If another condition is added to the subquery in the previous example, the subquery cannot be pulled up because the subquery references to the column in the outer query. Example:

  ```
  select * from t3 where t3.c1=(
         select t1.c1
         from t1 where c1 >(
               select max(t2.c1) from t2 where t2.c1=t1.c1 and t3.c1>t2.c2

  ));
  ```

– Pulling up a sublink in the **OR** clause

  If the **WHERE** condition contains an **EXIST** correlated sublink connected by **OR**:

  Example:

  ```
  select a, c from t1
  where t1.a = (select avg(a) from t3 where t1.b = t3.b) or
  exists (select * from t4 where t1.c = t4.c);
  ```

  The process of pulling up such a sublink is as follows:

  i. Extract **opExpr** from the **OR** clause in the **WHERE** condition. The value is **t1.a = (select avg(a) from t3 where t1.b = t3.b)**.

  ii. The **opExpr** contains a subquery. If the subquery can be pulled up, the subquery is rewritten as **select avg(a), t3.b from t3 group by t3.b**, generating the **NOT NULL** condition **t3.b is not null**. The **opExpr** is replaced with this **NOT NULL** condition. In this case, the SQL statement changes to:

  ```
  select a, c
  from t1 left join (select avg(a) avg, t3.b from t3 group by t3.b)  as t3 on (t1.a = avg
  and t1.b = t3.b)
  where t3.b is not null or exists (select * from t4 where t1.c = t4.c);
  ```

iii. Extract the **EXISTS** sublink **exists (select * from t4 where t1.c = t4.c)** from the **OR** clause to check whether the sublink can be pulled up. If it can be pulled up, it is converted into **select t4.c from t4 group by t4.c**, generating the **NOT NULL** condition **t4.c is not null**. In this case, the SQL statement changes to:

```
select t1.a, t1.c from t1 left join (select avg(a) avg, t3.b from t3 group by t3.b) as t3 on
(t1.a = avg and t1.b = t3.b) left join (select t5.c from t5 group by t5.c) as t5 on (t1.c =
t5.c) where t3.b is not null or t5.c is not null;
```

- **Sublink-release scenarios not supported by GaussDB**

  Except the sublinks described above, all the other sublinks cannot be pulled up. In this case, a join subquery is planned as the combination of subplans and broadcast. As a result, if tables in the subquery have a large amount of data, query performance may be poor.

  If a correlated subquery joins with two tables in outer queries, the subquery cannot be pulled up. You need to change the parent query into a **WITH** clause and then perform the join.

  Example:

```
select distinct t1.a, t2.a
from t1 left join t2 on t1.a=t2.a and not exists (select a,b from test1 where test1.a=t1.a and
test1.b=t2.a);
```

  The outer query is changed into:

```
with temp as
(
        select * from (select t1.a as a, t2.a as b from t1 left join t2 on t1.a=t2.a)

)
select distinct a,b
from temp
where not exists (select a,b from test1 where temp.a=test1.a and temp.b=test1.b);
```

  – The subquery (without **COUNT**) in the target list cannot be pulled up.

    Example:

```
explain (costs off)
select (select c2 from t2 where t1.c1 = t2.c1) ssq, t1.c2
from t1
where t1.c2 > 10;
```

    The execution plan is as follows:

```
explain (costs off)
select (select c2 from t2 where t1.c1 = t2.c1) ssq, t1.c2
from t1
where t1.c2 > 10;
          QUERY PLAN
--------------------------------
 Seq Scan on t1
   Filter: (c2 > 10)
   SubPlan 1
     ->  Seq Scan on t2
           Filter: (t1.c1 = c1)
(5 rows)
```

    The correlated subquery is displayed in the target list (query return list). Values need to be returned even if the condition **t1.c1=t2.c1** is not met. Therefore, use left outer join to join **T1** and **T2** so that SSQ can return padding values when the condition **t1.c1=t2.c1** is not met.

NOTE

ScalarSubQuery (SSQ) and Correlated-ScalarSubQuery (CSSQ) are described as follows:

- SSQ: a sublink that returns a scalar value of a single row with a single column
- CSSQ: an SSQ containing correlation conditions

The preceding SQL statement can be changed into:

```
with ssq as
(
    select t2.c2 from t2
)
select ssq.c2, t1.c2
from t1 left join ssq on t1.c1 = ssq.c2
where t1.c2 > 10;
```

The execution plan after the change is as follows:

```
          QUERY PLAN
---------------------------------
 Hash Right Join
   Hash Cond: (ssq.c2 = t1.c1)
   CTE ssq
     ->  Seq Scan on t2
   ->  CTE Scan on ssq
   ->  Hash
         ->  Seq Scan on t1
               Filter: (c2 > 10)
(8 rows)
```

In the preceding example, the SSQ in the target list is pulled up to right join, preventing poor performance caused by the plan involving subplans when the table (**T2**) in the subquery is too large.

– The subquery (with **COUNT**) in the target list cannot be pulled up.

Example:

```
select (select count(*) from t2 where t2.c1=t1.c1) cnt, t1.c1, t3.c1
from t1,t3
where t1.c1=t3.c1 order by cnt, t1.c1;
```

The execution plan is as follows:

```
                QUERY PLAN
-------------------------------------------
 Sort
   Sort Key: ((SubPlan 1)), t1.c1
   ->  Hash Join
         Hash Cond: (t1.c1 = t3.c1)
         ->  Seq Scan on t1
         ->  Hash
               ->  Seq Scan on t3
         SubPlan 1
           ->  Aggregate
                 ->  Seq Scan on t2
                       Filter: (c1 = t1.c1)
(11 rows)
```

The correlated subquery is displayed in the target list (query return list). Values need to be returned even if the condition **t1.c1=t2.c1** is not met. Therefore, use left outer join to join **T1** and **T2** so that SSQ can return padding values when the condition **t1.c1=t2.c1** is not met. However, **COUNT** is used, which requires that **0** is returned when the condition is not met. Therefore, **case-when NULL then 0 else count(*)** can be used.

The preceding SQL statement can be changed into:

```
with ssq as
(
    select count(*) cnt, c1 from t2 group by c1
)
select case when
            ssq.cnt is null then 0
            else ssq.cnt
       end cnt, t1.c1, t3.c1
from t1 left join ssq on ssq.c1 = t1.c1,t3
where t1.c1 = t3.c1
order by ssq.cnt, t1.c1;
```

The execution plan after the change is as follows:

```
              QUERY PLAN
-------------------------------------------
 Sort
   Sort Key: ssq.cnt, t1.c1
   CTE ssq
     ->  HashAggregate
           Group By Key: t2.c1
           ->  Seq Scan on t2
   ->  Hash Join
         Hash Cond: (t1.c1 = t3.c1)
         ->  Hash Left Join
               Hash Cond: (t1.c1 = ssq.c1)
               ->  Seq Scan on t1
               ->  Hash
                     ->  CTE Scan on ssq
         ->  Hash
               ->  Seq Scan on t3
(15 rows)
```

– Non-equivalent correlated subqueries cannot be pulled up.

  Example:

```
select t1.c1, t1.c2
from t1
where t1.c1 = (select agg() from t2.c2 > t1.c2);
```

  Non-equivalent correlated subqueries cannot be pulled up. You can perform join twice (one CorrelationKey and one rownum self-join) to rewrite the statement.

  You can rewrite the statement in either of the following ways:

  - Subquery rewriting

```
select t1.c1, t1.c2
from t1, (
    select t1.rowid, agg() aggref
    from t1,t2
    where t1.c2 > t2.c2 group by t1.rowid
) dt /* derived table */
where t1.rowid = dt.rowid AND t1.c1 = dt.aggref;
```

  - CTE rewriting

```
WITH dt as
(
    select t1.rowid, agg() aggref
    from t1,t2
    where t1.c2 > t2.c2 group by t1.rowid
)
select t1.c1, t1.c2
from t1, derived_table
where t1.rowid = derived_table.rowid AND
t1.c1 = derived_table.aggref;
```

NOTICE

- If the AGG type is **COUNT(*)**, **0** is used for data padding when **CASE-WHEN** is not matched. If the type is not **COUNT(*)**, **NULL** is used.
- CTE rewriting works better by using sharescan.

More Optimization Examples

Modify the **SELECT** statement by changing the subquery to a **JOIN** relationship between the primary table and the parent query or modifying the subquery to improve the query performance. Ensure that the subquery to be used is semantically correct.

```
explain (costs off) select * from t1 where t1.c1 in (select t2.c1 from t2 where t1.c1 = t2.c2);
           QUERY PLAN
--------------------------------
 Seq Scan on t1
   Filter: (SubPlan 1)
   SubPlan 1
     ->  Seq Scan on t2
           Filter: (t1.c1 = c2)
(5 rows)
```

In the preceding example, a subplan is used. To remove the subplan, you can modify the statement as follows:

```
explain (costs off) select * from t1 where exists (select t2.c1 from t2 where t1.c1 = t2.c2 and t1.c1 = t2.c1);
                QUERY PLAN
-----------------------------------------
 Hash Join
   Hash Cond: (t1.c1 = t2.c2)
   ->  Seq Scan on t1
   ->  Hash
         ->  HashAggregate
               Group By Key: t2.c2, t2.c1
               ->  Seq Scan on t2
                     Filter: (c2 = c1)
(8 rows)
```

In this way, the subplan is replaced by the hash-join between the two tables, greatly improving the execution efficiency.

9.3.6.3 Optimizing Statistics

Background

GaussDB generates optimal execution plans based on the cost estimation. Optimizers need to estimate the number of data rows and the cost based on statistics collected using **ANALYZE**. Therefore, the statistics is vital for the estimation of the number of rows and cost. Global statistics are collected using **ANALYZE**: **relpages** and **reltuples** in the **pg_class** table; **stadistinct**, **stanullfrac**, **stanumbersN**, **stavaluesN**, and **histogram_bounds** in the **pg_statistic** table.

Example 1: Poor Query Performance Due to the Lack of Statistics

In most cases, the lack of statistics about tables or columns involved in the query greatly affects the query performance.

The table structure is as follows:

```
CREATE TABLE LINEITEM
(
L_ORDERKEY         BIGINT        NOT NULL
, L_PARTKEY        BIGINT        NOT NULL
, L_SUPPKEY        BIGINT        NOT NULL
, L_LINENUMBER     BIGINT        NOT NULL
, L_QUANTITY       DECIMAL(15,2) NOT NULL
, L_EXTENDEDPRICE  DECIMAL(15,2) NOT NULL
, L_DISCOUNT       DECIMAL(15,2) NOT NULL
, L_TAX            DECIMAL(15,2) NOT NULL
, L_RETURNFLAG     CHAR(1)       NOT NULL
, L_LINESTATUS     CHAR(1)       NOT NULL
, L_SHIPDATE       DATE          NOT NULL
, L_COMMITDATE     DATE          NOT NULL
, L_RECEIPTDATE    DATE          NOT NULL
, L_SHIPINSTRUCT   CHAR(25)      NOT NULL
, L_SHIPMODE       CHAR(10)      NOT NULL
, L_COMMENT        VARCHAR(44)   NOT NULL
) with (orientation = column, COMPRESSION = MIDDLE);

CREATE TABLE ORDERS
(
O_ORDERKEY        BIGINT        NOT NULL
, O_CUSTKEY       BIGINT        NOT NULL
, O_ORDERSTATUS   CHAR(1)       NOT NULL
, O_TOTALPRICE    DECIMAL(15,2) NOT NULL
, O_ORDERDATE     DATE NOT NULL
, O_ORDERPRIORITY CHAR(15)      NOT NULL
, O_CLERK         CHAR(15)      NOT NULL
, O_SHIPPRIORITY  BIGINT        NOT NULL
, O_COMMENT       VARCHAR(79)   NOT NULL
)with (orientation = column, COMPRESSION = MIDDLE);
```

The query statements are as follows:

```
explain verbose select
count(*) as numwait
from
lineitem l1,
orders
where
o_orderkey = l1.l_orderkey
and o_orderstatus = 'F'
and l1.l_receiptdate > l1.l_commitdate
and not exists (
select
*
from
lineitem l3
where
l3.l_orderkey = l1.l_orderkey
and l3.l_suppkey <> l1.l_suppkey
and l3.l_receiptdate > l3.l_commitdate
)
order by
numwait desc;
```

If such an issue occurs, you can use the following methods to check whether statistics in tables or columns has been collected using **ANALYZE**.

1. Execute **EXPLAIN VERBOSE** to analyze the execution plan and check the warning information:

   ```
   WARNING:Statistics in some tables or columns(public.lineitem.l_receiptdate,
   public.lineitem.l_commitdate, public.lineitem.l_orderkey, public.lineitem.l_suppkey,
   public.orders.o_orderstatus, public.orders.o_orderkey) are not collected.
   HINT:Do analyze for them in order to generate optimized plan.
   ```

2. Check whether the following information exists in the log file in the **pg_log** directory. If it does, the poor query performance was caused by the lack of statistics in some tables or columns.

```
2017-06-14 17:28:30.336 CST 140644024579856 20971684 [BACKEND] LOG:Statistics in some tables or columns(public.lineitem.l_receiptdate, public.lineitem.l_commitdate, public.lineitem.l_orderkey, public.linei
tem.l_suppkey, public.orders.o_orderstatus, public.orders.o_orderkey) are not collected.
2017-06-14 17:28:30.336 CST 140644024579856 20971684 [BACKEND] HINT:Do analyze for them in order to generate optimized plan.
```

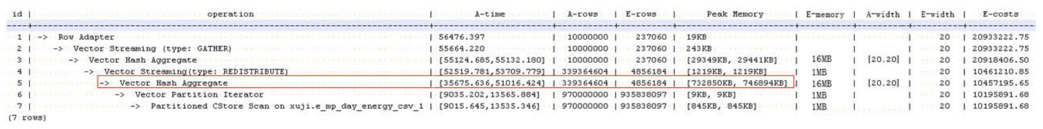
By using any of the preceding methods, you can identify tables or columns whose statistics have not been collected using **ANALYZE**. You can execute **ANALYZE** to warnings or tables and columns recorded in logs to resolve the problem.

9.3.6.4 Optimizing Operators

Background

A query statement needs to go through multiple operator procedures to generate the final result. Sometimes, the overall query performance deteriorates due to long execution time of certain operators, which are regarded as bottleneck operators. In this case, you need to execute the **EXPLAIN ANALYZE** or **EXPLAIN PERFORMANCE** command to view the bottleneck operators, and then perform optimization.

For example, in the following execution process, the execution time of the **Hashagg** operator accounts for about 66% [(51016-13535)/56476 ≈ 66%] of the total execution time. Therefore, the **Hashagg** operator is the bottleneck operator for this query. Optimize this operator first.



```
 id |                          operation                             |         A-time          |  A-rows   |  E-rows   |     Peak Memory        | E-memory | A-width | E-width |   E-costs
----+----------------------------------------------------------------+-------------------------+-----------+-----------+------------------------+----------+---------+---------+-------------
  1 | ->  Row Adapter                                                | 56476.397               |  10000000 |    237060 | 19KB                   |          |         |      20 | 20933222.75
  2 |    ->  Vector Streaming (type: GATHER)                         | 55664.220               |  10000000 |    237060 | 243KB                  |          |         |      20 | 20933222.75
  3 |       ->  Vector Hash Aggregate                                | [55124.685,55132.180]   |  10000000 |    237060 | [29349KB, 29441KB]     | 16MB     | [20,20] |      20 | 20918406.50
  4 |          ->  Vector Streaming(type: REDISTRIBUTE)              | [52519.781,53709.779]   | 339364604 |   4856184 | [1219KB, 1219KB]       | 1MB      |         |      20 | 10461210.85
  5 |             ->  Vector Hash Aggregate                          | [35675.636,51016.424]   | 339364604 |   4856184 | [732850KB, 746894KB]   | 16MB     | [20,20] |      20 | 10457195.65
  6 |                ->  Vector Partition Iterator                   | [9035.202,13565.884]    | 970000000 | 935838097 | [9KB, 9KB]             | 1MB      |         |      20 | 10195891.68
  7 |                   ->  Partitioned CStore Scan on xuji.e_mp_day_energy_csv_1 | [9015.645,13535.346] | 970000000 | 935838097 | [845KB, 845KB] | 1MB |    |      20 | 10195891.68
(7 rows)
```

Example

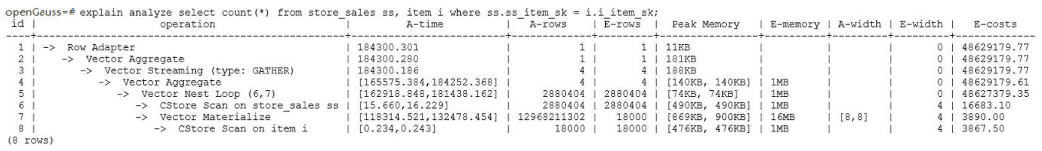
1. Scan the base table. For queries requiring large volume of data filtering, such as point queries or queries that need range scanning, a full table scan using SeqScan will take a long time. To facilitate scanning, you can create indexes on the condition column and select IndexScan for index scanning.

```
openGauss=#  explain (analyze on, costs off) select * from store_sales where ss_sold_date_sk = 2450944;
 id |             operation          |       A-time        | A-rows | Peak Memory  | A-width
----+--------------------------------+---------------------+--------+--------------+---------
  1 | ->  Streaming (type: GATHER)   | 3666.020            |   3360 | 195KB        |
  2 |    ->  Seq Scan on store_sales | [3594.611,3594.611] |   3360 | [34KB, 34KB] |
(2 rows)

 Predicate Information (identified by plan id)
-----------------------------------------------
   2 --Seq Scan on store_sales
         Filter: (ss_sold_date_sk = 2450944)
         Rows Removed by Filter: 4968936
openGauss=#  create index idx on store_sales_row(ss_sold_date_sk);
CREATE INDEX
openGauss=#  explain (analyze on, costs off) select * from store_sales_row where ss_sold_date_sk = 
2450944;
 id |                   operation                    |     A-time      | A-rows | Peak Memory  | A-width
----+------------------------------------------------+-----------------+--------+--------------+----------
```

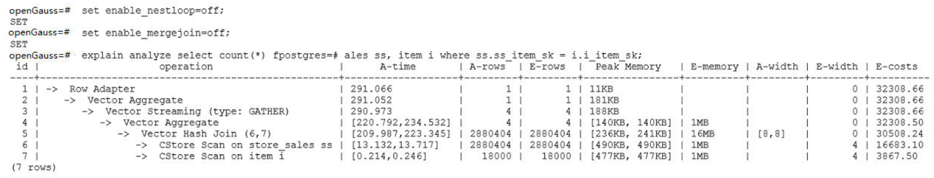
```
 1 | ->  Streaming (type: GATHER)                | 81.524          |   3360 | 195KB        |
 2 |    ->  Index Scan using idx on store_sales_row | [13.352,13.352] |   3360 | [34KB, 34KB] |
(2 rows)
```

In this example, the full table scan filters much data and returns 3360 records. After an index has been created on the **ss_sold_date_sk** column, the scanning efficiency is significantly boosted from 3.6s to 13 ms by using **IndexScan**.

2: If NestLoop is used for joining tables with a large number of rows, the join may take a long time. In the following example, NestLoop takes 181s. If **enable_mergejoin** is set to **off** to disable merge join and **enable_nestloop** is set to **off** to disable NestLoop so that the optimizer selects hash join, the join takes more than 200 ms.

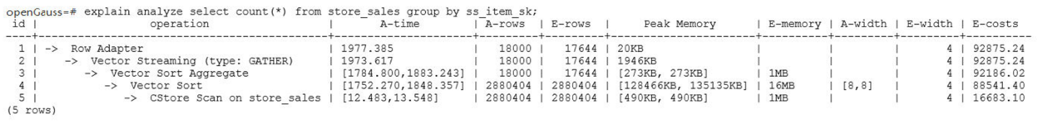


```
openGauss=# explain analyze select count(*) from store_sales ss, item i where ss.ss_item_sk = i.i_item_sk;
 id |                   operation                   |         A-time          |   A-rows    | E-rows  |  Peak Memory    | E-memory | A-width | E-width |   E-costs
----+-----------------------------------------------+-------------------------+-------------+---------+-----------------+----------+---------+---------+-------------
  1 | ->  Row Adapter                               | 184300.301              |           1 |       1 | 11KB            |          |         |       0 | 48629179.77
  2 |    ->  Vector Aggregate                       | 184300.280              |           1 |       1 | 181KB           |          |         |       0 | 48629179.77
  3 |       ->  Vector Streaming (type: GATHER)     | 184300.186              |           4 |       4 | 188KB           |          |         |       0 | 48629179.77
  4 |          ->  Vector Aggregate                 | [165575.384,184252.368] |           4 |       4 | [140KB, 140KB]  | 1MB      |         |       0 | 48629179.61
  5 |             ->  Vector Nest Loop (6,7)        | [162918.848,181438.162] |     2880404 | 2880404 | [74KB, 74KB]    | 1MB      |         |       0 | 48627379.35
  6 |                ->  CStore Scan on store_sales ss | [15.660,16.229]      |     2880404 | 2880404 | [490KB, 490KB]  | 1MB      |         |       4 | 16683.10
  7 |                ->  Vector Materialize         | [118314.521,132478.454] | 12968211302 |   18000 | [869KB, 900KB]  | 16MB     | [8,8]   |       4 | 3890.00
  8 |                   ->  CStore Scan on item i   | [0.234,0.243]           |       18000 |   18000 | [476KB, 476KB]  | 1MB      |         |       4 | 3867.50
(8 rows)
```

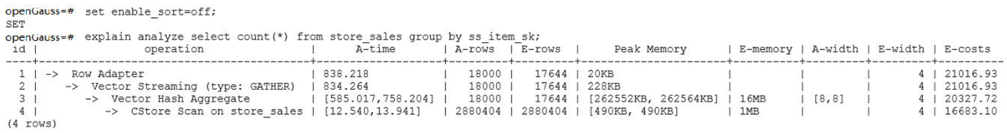


```
openGauss=#  set enable_nestloop=off;
SET
openGauss=#  set enable_mergejoin=off;
SET
openGauss=#  explain analyze select count(*) fpostgres=# ales ss, item i where ss.ss_item_sk = i.i_item_sk;
 id |                   operation                    |      A-time       | A-rows  | E-rows  |  Peak Memory    | E-memory | A-width | E-width | E-costs
----+------------------------------------------------+-------------------+---------+---------+-----------------+----------+---------+---------+----------
  1 | ->  Row Adapter                                | 291.066           |       1 |       1 | 11KB            |          |         |       0 | 32308.66
  2 |    ->  Vector Aggregate                        | 291.052           |       1 |       1 | 181KB           |          |         |       0 | 32308.66
  3 |       ->  Vector Streaming (type: GATHER)      | 290.973           |       4 |       4 | 188KB           |          |         |       0 | 32308.66
  4 |          ->  Vector Aggregate                  | [220.792,234.532] |       4 |       4 | [140KB, 140KB]  | 1MB      |         |       0 | 32308.50
  5 |             ->  Vector Hash Join (6,7)         | [209.987,223.345] | 2880404 | 2880404 | [236KB, 241KB]  | 16MB     | [8,8]   |       0 | 30508.24
  6 |                ->  CStore Scan on store_sales ss | [13.132,13.717] | 2880404 | 2880404 | [490KB, 490KB]  | 1MB      |         |       4 | 16683.10
  7 |                ->  CStore Scan on item i       | [0.214,0.246]     |   18000 |   18000 | [477KB, 477KB]  | 1MB      |         |       4 | 3867.50
(7 rows)
```

3. Generally, query performance can be improved by selecting **HashAgg**. If **Sort** and **GroupAgg** are used for a large result set, you need to set **enable_sort** to **off**. **HashAgg** consumes less time than **Sort** and **GroupAgg**.



```
openGauss=# explain analyze select count(*) from store_sales group by ss_item_sk;
 id |                 operation                  |        A-time         | A-rows  | E-rows  |      Peak Memory       | E-memory | A-width | E-width | E-costs
----+--------------------------------------------+-----------------------+---------+---------+------------------------+----------+---------+---------+----------
  1 | ->  Row Adapter                            | 1977.385              |   18000 |   17644 | 20KB                   |          |         |       4 | 92875.24
  2 |    ->  Vector Streaming (type: GATHER)     | 1973.617              |   18000 |   17644 | 1946KB                 |          |         |       4 | 92875.24
  3 |       ->  Vector Sort Aggregate            | [1784.800,1883.243]   |   18000 |   17644 | [273KB, 273KB]         | 1MB      |         |       4 | 92186.02
  4 |          ->  Vector Sort                   | [1752.270,1848.357]   | 2880404 | 2880404 | [128466KB, 135135KB]   | 16MB     | [8,8]   |       4 | 88541.40
  5 |             ->  CStore Scan on store_sales | [12.483,13.548]       | 2880404 | 2880404 | [490KB, 490KB]         | 1MB      |         |       4 | 16683.10
(5 rows)
```



```
openGauss=#  set enable_sort=off;
SET
openGauss=#  explain analyze select count(*) from store_sales group by ss_item_sk;
 id |                 operation                  |      A-time       | A-rows  | E-rows  |      Peak Memory       | E-memory | A-width | E-width | E-costs
----+--------------------------------------------+-------------------+---------+---------+------------------------+----------+---------+---------+----------
  1 | ->  Row Adapter                            | 838.218           |   18000 |   17644 | 20KB                   |          |         |       4 | 21016.93
  2 |    ->  Vector Streaming (type: GATHER)     | 834.264           |   18000 |   17644 | 228KB                  |          |         |       4 | 21016.93
  3 |       ->  Vector Hash Aggregate            | [585.017,758.204] |   18000 |   17644 | [262552KB, 262564KB]   | 16MB     | [8,8]   |       4 | 20327.72
  4 |          ->  CStore Scan on store_sales    | [12.540,13.941]   | 2880404 | 2880404 | [490KB, 490KB]         | 1MB      |         |       4 | 16683.10
(4 rows)
```

9.3.7 Experience in Rewriting SQL Statements

Based on the SQL execution mechanism and a large number of practices, SQL statements can be optimized by following certain rules to enable the database to execute SQL statements more quickly and obtain correct results. You can comply with these rules to improve service query efficiency.

- Replace **UNION** with **UNION ALL**.

  UNION eliminates duplicate rows while merging two result sets but **UNION ALL** merges the two result sets without deduplication. Therefore, replace **UNION** with **UNION ALL** if you are sure that the two result sets do not contain duplicate rows based on the service logic.

- Add **NOT NULL** to the join columns.

If there are many NULL values in the **JOIN** columns, you can add the filter criterion **IS NOT NULL** to filter data in advance to improve the **JOIN** efficiency.

- Convert **NOT IN** to **NOT EXISTS**.

  nestloop anti join must be used to implement **NOT IN**, and **hash anti join** is required for **NOT EXISTS**. If no NULL value exists in the **JOIN** columns, **NOT IN** is equivalent to **NOT EXISTS**. Therefore, if you are sure that no NULL value exists, you can convert **NOT IN** to **NOT EXISTS** to generate **hash join** and to improve the query performance.

  As shown in the following statement, the **t2.d2** column does not contain null values (it is set to **NOT NULL**) and **NOT EXISTS** is used for the query.

  ```
  SELECT * FROM t1 WHERE  NOT EXISTS (SELECT * FROM t2 WHERE t1.c1=t2.d2);
  ```

  The generated execution plan is as follows:

  ```
  QUERY PLAN
  ------------------------------
  Hash Anti Join
  Hash Cond: (t1.c1 = t2.d2)
  ->  Seq Scan on t1
  ->  Hash
  ->  Seq Scan on t2
  (5 rows)
  ```

- Use **hashagg**.

  If a plan involving groupAgg and SORT operations generated by the **GROUP BY** statement is poor in performance, you can set **work_mem** to a larger value to generate a **hashagg** plan, which does not require sorting and improves the performance.

- Replace functions with **CASE** statements.

  The GaussDB performance greatly deteriorates if a large number of functions are called. In this case, you can change the pushdown functions to **CASE** statements.

- Do not use functions or expressions for indexes.

  Using functions or expressions for indexes stops indexing. Instead, it enables scanning on the full table.

- Do not use **!=** or **<>** operators, **NULL**, **OR**, or implicit parameter conversion in **WHERE** clauses.

- Split complex SQL statements.

  You can split an SQL statement into several ones and save the execution result to a temporary table if the SQL statement is too complex to be tuned using the solutions above, including but not limited to the following scenarios:

  – The same subquery is involved in multiple SQL statements of a job and the subquery contains large amounts of data.

  – Incorrect plan cost causes a small hash bucket of subquery. For example, the actual number of rows is 10 million, but only 1000 rows are in hash bucket.

  – Functions such as **substr** and **to_number** cause incorrect measures for subqueries containing large amounts of data.

9.3.8 Resetting Key Parameters During SQL Tuning

This section introduces key parameters of the primary database node that affect optimization of SQL statements in GaussDB. For details about the parameter configurations, see **Configuring Running Parameters**.

Table 9-3 Parameters of the primary database node

Parameter/ Reference Value	Description
enable_nestloop=on	Specifies how the optimizer uses **Nest Loop Join**. If this parameter is set to **on**, the optimizer preferentially uses **Nest Loop Join**. If it is set to **off**, the optimizer preferentially uses other methods, if any. **NOTE** If you only want to temporarily change the value of this parameter during the current database connection (that is, the current session), execute the following SQL statement: `SET enable_nestloop to off;` By default, this parameter is set to **on**. Change the value as required. Generally, nested loop join has the poorest performance among the three **JOIN** methods (nested loop join, merge join, and hash join). You are advised to set this parameter to **off**.
enable_bitmapscan=on	Specifies whether the optimizer uses bitmap scanning. If the value is **on**, bitmap scanning is used. If the value is **off**, it is not used. **NOTE** If you only want to temporarily change the value of this parameter during the current database connection (that is, the current session), execute the following SQL statement: `SET enable_bitmapscan to off;` The bitmap scanning applies only in the query condition where **a > 1 and b > 1** and indexes are created on columns **a** and **b**. During performance tuning, if the query performance is poor and bitmapscan operators are in the execution plan, set this parameter to **off** and check whether the performance is improved.
enable_hashagg=on	Specifies whether to enable the optimizer's use of Hash-aggregation plan types.
enable_hashjoin=on	Specifies whether to enable the optimizer's use of Hash-join plan types.
enable_mergejoin=on	Specifies whether to enable the optimizer's use of Hash-merge plan types.
enable_indexscan=on	Specifies whether to enable the optimizer's use of index-scan plan types.
enable_indexonlyscan=on	Specifies whether to enable the optimizer's use of index-only-scan plan types.

Parameter/ Reference Value	Description
enable_seqscan=on	Specifies whether the optimizer uses bitmap scanning. It is impossible to suppress sequential scans entirely, but setting this variable to **off** encourages the optimizer to choose other methods if available.
enable_sort=on	Specifies the optimizer sorts. It is impossible to fully suppress explicit sorts, but setting this variable to **off** allows the optimizer to preferentially choose other methods if available.
rewrite_rule	Specifies whether the optimizer enables the **LAZY_AGG** and **MAGIC_SET** rewriting rules.
sql_beta_feature	Specifies whether the optimizer enables the **SEL_SEMI_POISSON**, **SEL_EXPR_INSTR**, **PARAM_PATH_GEN**, **RAND_COST_OPT**, **PARAM_PATH_OPT**, **PAGE_EST_OPT**, **CANONICAL_PATHKEY**, **PARTITION_OPFUSION**, **PREDPUSH_SAME_LEVEL**, **PARTITION_FDW_ON**, and **DISABLE_BITMAP_COST_WITH_LOSSY_PAGES** beta features.

9.3.9 Hint-based Tuning

9.3.9.1 Plan Hint Optimization

In plan hints, you can specify a join order, join and scan operations, and the number of rows in a result to tune an execution plan, improving query performance.

GaussDB also provides the SQL patch function. You can create an SQL patch to make hints take effect without modifying service statements. For details, see "Maintainability > SQL PATCH" in *Feature Description*.

Function

Plan hints are specified in the following format after keywords such as SELECT, INSERT, UPDATE, DELETE, and MERGE:

```
/*+ <plan hint>*/
```

You can specify multiple hints for a query plan and separate them by spaces. A hint specified for a query plan does not apply to its subquery plans. To specify a hint for a subquery, add the hint following the **SELECT** of this subquery.

Example:

```
select /*+ <plan_hint1> <plan_hint2> */ * from t1, (select /*+ <plan_hint3> */ * from t2) where 1=1;
```

In the preceding command, <*plan_hint1*> and <*plan_hint2*> are the hints of a query, and <*plan_hint3*> is the hint of its subquery.

NOTICE

If a hint is specified in the **CREATE VIEW** statement, the hint will be applied each time this view is used.

If the random plan function is enabled (**plan_mode_seed** is set to a value other than 0), the specified hint will not be used.

Scope

Currently, the following hints are supported:

- Join order hints (**leading**)
- Join operation hints, excluding the **semi join**, **anti join**, and **unique plan** hints
- Rows hints
- Scan operation hints, supporting only **tablescan**, **indexscan**, and **indexonlyscan**
- Sublink name hints

Precautions

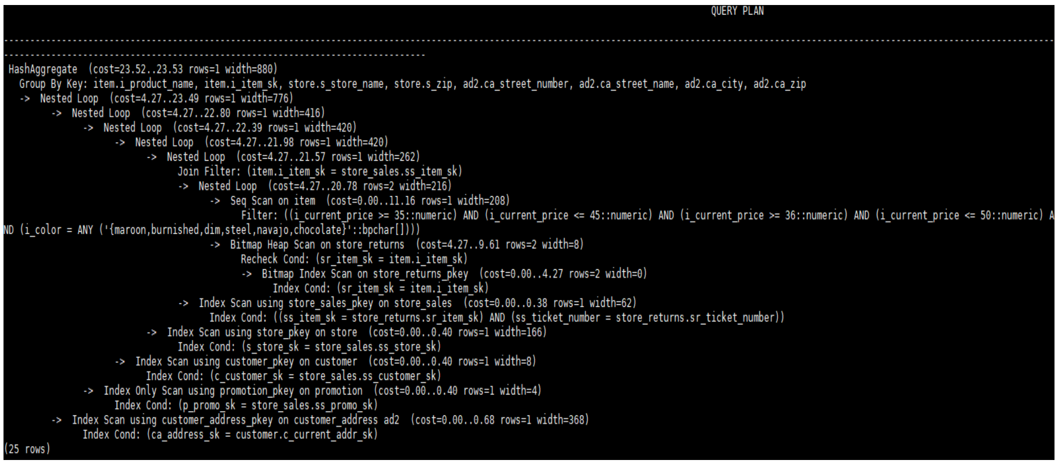
Hints do not support **Agg**, **Sort**, **Setop**, or **Subplan**.

Examples

The following is the original plan and is used for comparing with the optimized ones:

```
create table store
(
    s_store_sk                integer               not null,
    s_store_id                char(16)              not null,
    s_rec_start_date          date                          ,
    s_rec_end_date            date                          ,
    s_closed_date_sk          integer                       ,
    s_store_name              varchar(50)                   ,
    s_number_employees        integer                       ,
    s_floor_space             integer                       ,
    s_hours                   char(20)                      ,
    s_manager                 varchar(40)                   ,
    s_market_id               integer                       ,
    s_geography_class         varchar(100)                  ,
    s_market_desc             varchar(100)                  ,
    s_market_manager          varchar(40)                   ,
    s_division_id             integer                       ,
    s_division_name           varchar(50)                   ,
    s_company_id              integer                       ,
    s_company_name            varchar(50)                   ,
    s_street_number           varchar(10)                   ,
    s_street_name             varchar(60)                   ,
    s_street_type             char(15)                      ,
    s_suite_number            char(10)                      ,
    s_city                    varchar(60)                   ,
    s_county                  varchar(30)                   ,
    s_state                   char(2)                       ,
    s_zip                     char(10)                      ,
    s_country                 varchar(20)                   ,
    s_gmt_offset              decimal(5,2)                  ,
    s_tax_precentage          decimal(5,2)                  ,
    primary key (s_store_sk)
```

```
);
create table store_sales
(
    ss_sold_date_sk           integer                       ,
    ss_sold_time_sk           integer                       ,
    ss_item_sk                integer               not null,
    ss_customer_sk            integer                       ,
    ss_cdemo_sk               integer                       ,
    ss_hdemo_sk               integer                       ,
    ss_addr_sk                integer                       ,
    ss_store_sk               integer                       ,
    ss_promo_sk               integer                       ,
    ss_ticket_number          integer               not null,
    ss_quantity               integer                       ,
    ss_wholesale_cost         decimal(7,2)                  ,
    ss_list_price             decimal(7,2)                  ,
    ss_sales_price            decimal(7,2)                  ,
    ss_ext_discount_amt       decimal(7,2)                  ,
    ss_ext_sales_price        decimal(7,2)                  ,
    ss_ext_wholesale_cost     decimal(7,2)                  ,
    ss_ext_list_price         decimal(7,2)                  ,
    ss_ext_tax                decimal(7,2)                  ,
    ss_coupon_amt             decimal(7,2)                  ,
    ss_net_paid               decimal(7,2)                  ,
    ss_net_paid_inc_tax       decimal(7,2)                  ,
    ss_net_profit             decimal(7,2)                  ,
    primary key (ss_item_sk, ss_ticket_number)
);
create table store_returns
(
    sr_returned_date_sk       integer                       ,
    sr_return_time_sk         integer                       ,
    sr_item_sk                integer               not null,
    sr_customer_sk            integer                       ,
    sr_cdemo_sk               integer                       ,
    sr_hdemo_sk               integer                       ,
    sr_addr_sk                integer                       ,
    sr_store_sk               integer                       ,
    sr_reason_sk              integer                       ,
    sr_ticket_number          integer               not null,
    sr_return_quantity        integer                       ,
    sr_return_amt             decimal(7,2)                  ,
    sr_return_tax             decimal(7,2)                  ,
    sr_return_amt_inc_tax     decimal(7,2)                  ,
    sr_fee                    decimal(7,2)                  ,
    sr_return_ship_cost       decimal(7,2)                  ,
    sr_refunded_cash          decimal(7,2)                  ,
    sr_reversed_charge        decimal(7,2)                  ,
    sr_store_credit           decimal(7,2)                  ,
    sr_net_loss               decimal(7,2)                  ,
    primary key (sr_item_sk, sr_ticket_number)
);
create table customer
(
    c_customer_sk             integer               not null,
    c_customer_id             char(16)              not null,
    c_current_cdemo_sk        integer                       ,
    c_current_hdemo_sk        integer                       ,
    c_current_addr_sk         integer                       ,
    c_first_shipto_date_sk    integer                       ,
    c_first_sales_date_sk     integer                       ,
    c_salutation              char(10)                      ,
    c_first_name              char(20)                      ,
    c_last_name               char(30)                      ,
    c_preferred_cust_flag     char(1)                       ,
    c_birth_day               integer                       ,
    c_birth_month             integer                       ,
    c_birth_year              integer                       ,
    c_birth_country           varchar(20)                   ,
    c_login                  char(13)                  ,
    c_email_address          char(50)                  ,
    c_last_review_date       char(10)                  ,
    primary key (c_customer_sk)
);
create table promotion
(
    p_promo_sk               integer               not null,
    p_promo_id               char(16)              not null,
    p_start_date_sk          integer                   ,
    p_end_date_sk            integer                   ,
    p_item_sk                integer                   ,
    p_cost                   decimal(15,2)             ,
    p_response_target        integer                   ,
    p_promo_name             char(50)                  ,
    p_channel_dmail          char(1)                   ,
    p_channel_email          char(1)                   ,
    p_channel_catalog        char(1)                   ,
    p_channel_tv             char(1)                   ,
    p_channel_radio          char(1)                   ,
    p_channel_press          char(1)                   ,
    p_channel_event          char(1)                   ,
    p_channel_demo           char(1)                   ,
    p_channel_details        varchar(100)              ,
    p_purpose                char(15)                  ,
    p_discount_active        char(1)                   ,
    primary key (p_promo_sk)
);
create table customer_address
(
    ca_address_sk            integer               not null,
    ca_address_id            char(16)              not null,
    ca_street_number         char(10)                  ,
    ca_street_name           varchar(60)               ,
    ca_street_type           char(15)                  ,
    ca_suite_number          char(10)                  ,
    ca_city                  varchar(60)               ,
    ca_county                varchar(30)               ,
    ca_state                 char(2)                   ,
    ca_zip                   char(10)                  ,
    ca_country               varchar(20)               ,
    ca_gmt_offset            decimal(5,2)              ,
    ca_location_type         char(20)                  ,
    primary key (ca_address_sk)
);
create table item
(
    i_item_sk                integer               not null,
    i_item_id                char(16)              not null,
    i_rec_start_date         date                      ,
    i_rec_end_date           date                      ,
    i_item_desc              varchar(200)              ,
    i_current_price          decimal(7,2)              ,
    i_wholesale_cost         decimal(7,2)              ,
    i_brand_id               integer                   ,
    i_brand                  char(50)                  ,
    i_class_id               integer                   ,
    i_class                  char(50)                  ,
    i_category_id            integer                   ,
    i_category               char(50)                  ,
    i_manufact_id            integer                   ,
    i_manufact               char(50)                  ,
    i_size                   char(20)                  ,
    i_formulation            char(20)                  ,
    i_color                  char(20)                  ,
    i_units                  char(10)                  ,
    i_container              char(10)                  ,
    i_manager_id             integer                   ,
    i_product_name           char(50)                  ,
    primary key (i_item_sk)
);
explain
select i_product_name product_name
,i_item_sk item_sk
,s_store_name store_name
,s_zip store_zip
,ad2.ca_street_number c_street_number
,ad2.ca_street_name c_street_name
,ad2.ca_city c_city
,ad2.ca_zip c_zip
,count(*) cnt
,sum(ss_wholesale_cost) s1
,sum(ss_list_price) s2
,sum(ss_coupon_amt) s3
FROM   store_sales
,store_returns
,store
,customer
,promotion
,customer_address ad2
,item
WHERE  ss_store_sk = s_store_sk AND
ss_customer_sk = c_customer_sk AND
ss_item_sk = i_item_sk and
ss_item_sk = sr_item_sk and
ss_ticket_number = sr_ticket_number and
c_current_addr_sk = ad2.ca_address_sk and
ss_promo_sk = p_promo_sk and
i_color in ('maroon','burnished','dim','steel','navajo','chocolate') and
i_current_price between 35 and 35 + 10 and
i_current_price between 35 + 1 and 35 + 15
group by i_product_name
,i_item_sk
,s_store_name
,s_zip
,ad2.ca_street_number
,ad2.ca_street_name
,ad2.ca_city
,ad2.ca_zip
;
```



```
                                                                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------
 HashAggregate  (cost=23.52..23.53 rows=1 width=880)
   Group By Key: item.i_product_name, item.i_item_sk, store.s_store_name, store.s_zip, ad2.ca_street_number, ad2.ca_street_name, ad2.ca_city, ad2.ca_zip
   ->  Nested Loop  (cost=4.27..23.49 rows=1 width=776)
         ->  Nested Loop  (cost=4.27..22.80 rows=1 width=416)
               ->  Nested Loop  (cost=4.27..22.39 rows=1 width=420)
                     ->  Nested Loop  (cost=4.27..21.98 rows=1 width=420)
                           ->  Nested Loop  (cost=4.27..21.57 rows=1 width=262)
                                 Join Filter: (item.i_item_sk = store_sales.ss_item_sk)
                                 ->  Nested Loop  (cost=4.27..20.78 rows=2 width=216)
                                       ->  Seq Scan on item  (cost=0.00..11.16 rows=1 width=208)
                                             Filter: ((i_current_price >= 35::numeric) AND (i_current_price <= 45::numeric) AND (i_current_price >= 36::numeric) AND (i_current_price <= 50::numeric) A
ND (i_color = ANY ('{maroon,burnished,dim,steel,navajo,chocolate}'::bpchar[])))
                                       ->  Bitmap Heap Scan on store_returns  (cost=4.27..9.61 rows=2 width=8)
                                             Recheck Cond: (sr_item_sk = item.i_item_sk)
                                             ->  Bitmap Index Scan on store_returns_pkey  (cost=0.00..4.27 rows=2 width=0)
                                                   Index Cond: (sr_item_sk = item.i_item_sk)
                                 ->  Index Scan using store_sales_pkey on store_sales  (cost=0.00..0.38 rows=1 width=62)
                                       Index Cond: ((ss_item_sk = store_returns.sr_item_sk) AND (ss_ticket_number = store_returns.sr_ticket_number))
                           ->  Index Scan using store_pkey on store  (cost=0.00..0.40 rows=1 width=166)
                                 Index Cond: (s_store_sk = store_sales.ss_store_sk)
                     ->  Index Scan using customer_pkey on customer  (cost=0.00..0.40 rows=1 width=8)
                           Index Cond: (c_customer_sk = store_sales.ss_customer_sk)
               ->  Index Only Scan using promotion_pkey on promotion  (cost=0.00..0.40 rows=1 width=4)
                     Index Cond: (p_promo_sk = store_sales.ss_promo_sk)
         ->  Index Scan using customer_address_pkey on customer_address ad2  (cost=0.00..0.68 rows=1 width=368)
               Index Cond: (ca_address_sk = customer.c_current_addr_sk)
(25 rows)
```

9.3.9.2 Join Order Hints

Function

Theses hints specify the join order and outer/inner tables.

Syntax

- Specify only the join order.

```
leading(join_table_list)
```

- Specify the join order and outer/inner tables. The outer/inner tables are specified by the outermost parentheses.

```
leading((join_table_list))
```

Parameter Description

join_table_list specifies the tables to be joined. The values can be table names or table aliases. If a subquery is pulled up, the value can also be the subquery alias. Separate the values with spaces. You can add parentheses to specify the join priorities of tables.

NOTICE

A table name or alias can only be a string without a schema name.

An alias (if any) is used to represent a table.

To prevent semantic errors, tables in the list must meet the following requirements:

- The tables must exist in the query or its subquery to be pulled up.
- The table names must be unique in the query or subquery to be pulled up. If they are not, their aliases must be unique.
- A table appears only once in the list.
- An alias (if any) is used to represent a table.

For example:

leading(t1 t2 t3 t4 t5): **t1**, **t2**, **t3**, **t4**, and **t5** are joined. The join order and outer/inner tables are not specified.

leading((t1 t2 t3 t4 t5)): **t1**, **t2**, **t3**, **t4**, and **t5** are joined in sequence. The table on the right is used as the inner table in each join.

leading(t1 (t2 t3 t4) t5): First, **t2**, **t3**, and **t4** are joined and the outer/inner tables are not specified. Then, the result is joined with **t1** and **t5**, and the outer/inner tables are not specified.

leading((t1 (t2 t3 t4) t5)): First, **t2**, **t3**, and **t4** are joined and the outer/inner tables are not specified. Then, the result is joined with **t1**, and **(t2 t3 t4)** is used as the inner table. Finally, the result is joined with **t5**, and **t5** is used as the inner table.

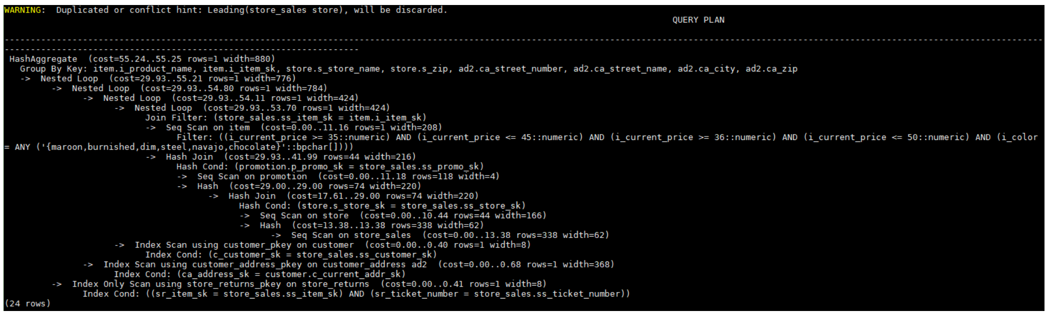
leading((t1 (t2 t3) t4 t5)) leading((t3 t2)): First, **t2** and **t3** are joined and **t2** is used as the inner table. Then, the result is joined with **t1**, and **(t2 t3)** is used as the inner table. Finally, the result is joined with **t4** and then **t5**, and the table on the right in each join is used as the inner table.

Example

Hint the query plan in **Examples** as follows:

```
explain
select /*+ leading((((((store_sales store) promotion) item) customer) ad2) store_returns) leading((store
store_sales))*/ i_product_name product_name ...
```

First, **store_sales** and **store** are joined and **store_sales** is the inner table. Then, the result is joined with **promotion**, **item**, **customer**, **ad2**, and **store_returns** in sequence. The optimized plan is as follows:



```
WARNING:  Duplicated or conflict hint: Leading(store_sales store), will be discarded.
                                                                                          QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------
 HashAggregate  (cost=55.24..55.25 rows=1 width=880)
   Group By Key: item.i_product_name, item.i_item_sk, store.s_store_name, store.s_zip, ad2.ca_street_number, ad2.ca_street_name, ad2.ca_city, ad2.ca_zip
   ->  Nested Loop  (cost=29.93..55.21 rows=1 width=776)
         ->  Nested Loop  (cost=29.93..54.80 rows=1 width=784)
               ->  Nested Loop  (cost=29.93..54.11 rows=1 width=424)
                     ->  Nested Loop  (cost=29.93..53.70 rows=1 width=424)
                           Join Filter: (store_sales.ss_item_sk = item.i_item_sk)
                           ->  Seq Scan on item  (cost=0.00..11.16 rows=1 width=208)
                                 Filter: ((i_current_price >= 35::numeric) AND (i_current_price <= 45::numeric) AND (i_current_price >= 36::numeric) AND (i_current_price <= 50::numeric) AND (i_color
= ANY ('{maroon,burnished,dim,steel,navajo,chocolate}'::bpchar[])))
                           ->  Hash Join  (cost=29.93..41.99 rows=44 width=216)
                                 Hash Cond: (promotion.p_promo_sk = store_sales.ss_promo_sk)
                                 ->  Seq Scan on promotion  (cost=0.00..11.18 rows=118 width=4)
                                 ->  Hash  (cost=29.00..29.00 rows=74 width=220)
                                       ->  Hash Join  (cost=17.61..29.00 rows=74 width=220)
                                             Hash Cond: (store.s_store_sk = store_sales.ss_store_sk)
                                             ->  Seq Scan on store  (cost=0.00..10.44 rows=44 width=166)
                                             ->  Hash  (cost=13.38..13.38 rows=338 width=62)
                                                   ->  Seq Scan on store_sales  (cost=0.00..13.38 rows=338 width=62)
                     ->  Index Scan using customer_pkey on customer  (cost=0.00..0.40 rows=1 width=8)
                           Index Cond: (c_customer_sk = store_sales.ss_customer_sk)
               ->  Index Scan using customer_address_pkey on customer_address ad2  (cost=0.00..0.68 rows=1 width=368)
                     Index Cond: (ca_address_sk = customer.c_current_addr_sk)
         ->  Index Only Scan using store_returns_pkey on store_returns  (cost=0.00..0.41 rows=1 width=8)
               Index Cond: ((sr_item_sk = store_sales.ss_item_sk) AND (sr_ticket_number = store_sales.ss_ticket_number))
(24 rows)
```

For details about the warning at the top of the plan, see **Hint Errors, Conflicts, and Other Warnings**.

9.3.9.3 Join Operation Hints

Function

These hints specify the join method, which can be nested loop join, hash join, or merge join.

Syntax

```
[no] nestloop|hashjoin|mergejoin(table_list)
```

Parameter Description

- **no** indicates that the specified hint will not be used for a join.
- **table_list** specifies the tables to be joined. The values are the same as those of **join_table_list** but contain no parentheses.

For example:

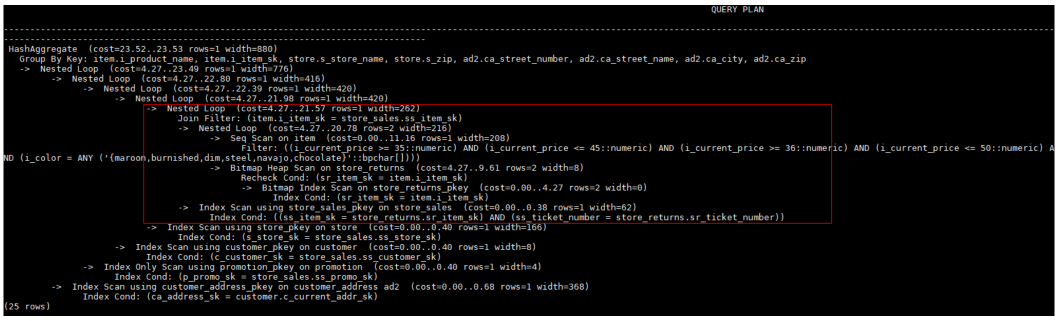
no nestloop(t1 t2 t3): **nestloop** is not used for joining **t1**, **t2**, and **t3**. The three tables may be joined in either of the two ways: Join **t2** and **t3**, and then **t1**; join **t1** and **t2**, and then **t3**. This hint takes effect only for the last join. If necessary, you can hint other joins. For example, you can add **no nestloop(t2 t3)** to join **t2** and **t3** first and to forbid the use of **nestloop**.

Example

Hint the query plan in Example as follows:

```
explain
select /*+ nestloop(store_sales store_returns item) */ i_product_name product_name ...
```

nestloop is used for the last join between **store_sales**, **store_returns**, and **item**. The optimized plan is as follows:



```
                                                                              QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------
 HashAggregate  (cost=23.52..23.53 rows=1 width=880)
   Group By Key: item.i_product_name, item.i_item_sk, store.s_store_name, store.s_zip, ad2.ca_street_number, ad2.ca_street_name, ad2.ca_city, ad2.ca_zip
   ->  Nested Loop  (cost=4.27..23.49 rows=1 width=776)
         ->  Nested Loop  (cost=4.27..22.80 rows=1 width=416)
               ->  Nested Loop  (cost=4.27..22.39 rows=1 width=420)
                     ->  Nested Loop  (cost=4.27..21.98 rows=1 width=420)
                           ->  Nested Loop  (cost=4.27..21.57 rows=1 width=262)
                                 Join Filter: (item.i_item_sk = store_sales.ss_item_sk)
                                 ->  Nested Loop  (cost=4.27..20.78 rows=2 width=216)
                                       ->  Seq Scan on item  (cost=0.00..11.16 rows=1 width=208)
                                             Filter: ((i_current_price >= 35::numeric) AND (i_current_price <= 45::numeric) AND (i_current_price >= 36::numeric) AND (i_current_price <= 50::numeric) A
ND (i_color = ANY ('{maroon,burnished,dim,steel,navajo,chocolate}'::bpchar[])))
                                       ->  Bitmap Heap Scan on store_returns  (cost=4.27..9.61 rows=2 width=8)
                                             Recheck Cond: (sr_item_sk = item.i_item_sk)
                                             ->  Bitmap Index Scan on store_returns_pkey  (cost=0.00..4.27 rows=2 width=0)
                                                   Index Cond: (sr_item_sk = item.i_item_sk)
                                 ->  Index Scan using store_sales_pkey on store_sales  (cost=0.00..0.38 rows=1 width=62)
                                       Index Cond: ((ss_item_sk = store_returns.sr_item_sk) AND (ss_ticket_number = store_returns.sr_ticket_number))
                           ->  Index Scan using store_pkey on store  (cost=0.00..0.40 rows=1 width=166)
                                 Index Cond: (s_store_sk = store_sales.ss_store_sk)
                     ->  Index Scan using customer_pkey on customer  (cost=0.00..0.40 rows=1 width=8)
                           Index Cond: (c_customer_sk = store_sales.ss_customer_sk)
               ->  Index Only Scan using promotion_pkey on promotion  (cost=0.00..0.40 rows=1 width=4)
                     Index Cond: (p_promo_sk = store_sales.ss_promo_sk)
         ->  Index Scan using customer_address_pkey on customer_address ad2  (cost=0.00..0.68 rows=1 width=368)
               Index Cond: (ca_address_sk = customer.c_current_addr_sk)
(25 rows)
```

9.3.9.4 Rows Hints

Function

These hints specify the number of rows in an intermediate result set. Both absolute values and relative values are supported.

Syntax

```
rows(table_list #|+|-|* const)
```

Parameter Description

- **#**, **+**, **-**, and ***** are operators used for hinting the estimation. **#** indicates that the original estimation is used without any calculation. **+**, **-**, and ***** indicate that the original estimation is calculated using these operators. The minimum calculation result is 1. *table_list* specifies the tables to be joined. The values are the same as those of **table_list** in **Join Operation Hints**.
- *const* can be any non-negative number and supports scientific notation.

Example:

rows(t1 #5): The result set of **t1** is five rows.

rows(t1 t2 t3 *1000): Multiply the result set of joined **t1**, **t2**, and **t3** by 1000.

Suggestion

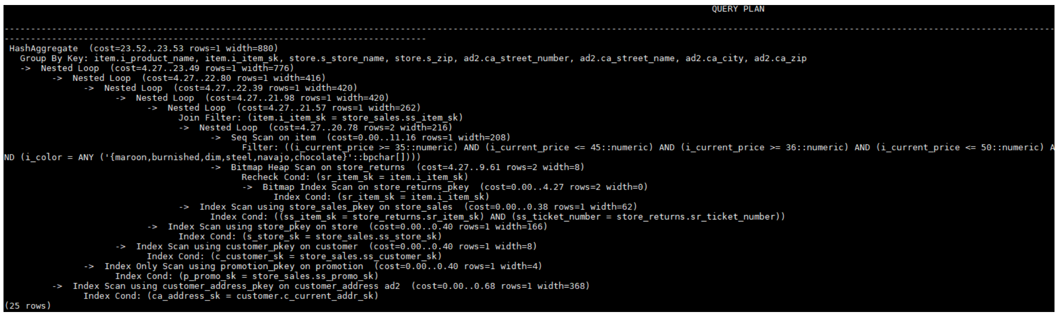
- The hint using ***** for two tables is recommended. This hint will be triggered if the two tables appear on two sides of a join. For example, if the hint is **rows(t1 t2 * 3)**, the join result of **(t1 t3 t4)** and **(t2 t5 t6)** will be multiplied by 3 because **t1** and **t2** appear on both sides of the join.
- **rows** hints can be specified for the result sets of a single table, multiple tables, function tables, and subquery scan tables.

Examples

Hint the query plan in **Examples** as follows:

```
explain
select /*+ rows(store_sales store_returns *50) */ i_product_name product_name ...
```

Multiply the result set of joined **store_sales** and **store_returns** by 50. The optimized plan is as follows:



```
                                                                                                                             QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------
 HashAggregate  (cost=23.52..23.53 rows=1 width=880)
   Group By Key: item.i_product_name, item.i_item_sk, store.s_store_name, store.s_zip, ad2.ca_street_number, ad2.ca_street_name, ad2.ca_city, ad2.ca_zip
   ->  Nested Loop  (cost=4.27..23.49 rows=1 width=776)
         ->  Nested Loop  (cost=4.27..22.80 rows=1 width=416)
               ->  Nested Loop  (cost=4.27..22.39 rows=1 width=420)
                     ->  Nested Loop  (cost=4.27..21.98 rows=1 width=420)
                           ->  Nested Loop  (cost=4.27..21.57 rows=1 width=262)
                                 Join Filter: (item.i_item_sk = store_sales.ss_item_sk)
                                 ->  Nested Loop  (cost=4.27..20.78 rows=2 width=216)
                                       ->  Seq Scan on item  (cost=0.00..11.16 rows=1 width=208)
                                             Filter: ((i_current_price >= 35::numeric) AND (i_current_price <= 45::numeric) AND (i_current_price >= 36::numeric) AND (i_current_price <= 50::numeric) A
ND (i_color = ANY ('{maroon,burnished,dim,steel,navajo,chocolate}'::bpchar[])))
                                       ->  Bitmap Heap Scan on store_returns  (cost=4.27..9.61 rows=2 width=8)
                                             Recheck Cond: (sr_item_sk = item.i_item_sk)
                                             ->  Bitmap Index Scan on store_returns_pkey  (cost=0.00..4.27 rows=2 width=0)
                                                   Index Cond: (sr_item_sk = item.i_item_sk)
                                 ->  Index Scan using store_sales_pkey on store_sales  (cost=0.00..0.38 rows=1 width=62)
                                       Index Cond: ((ss_item_sk = store_returns.sr_item_sk) AND (ss_ticket_number = store_returns.sr_ticket_number))
                           ->  Index Scan using store_pkey on store  (cost=0.00..0.40 rows=1 width=166)
                                 Index Cond: (s_store_sk = store_sales.ss_store_sk)
                     ->  Index Scan using customer_pkey on customer  (cost=0.00..0.40 rows=1 width=8)
                           Index Cond: (c_customer_sk = store_sales.ss_customer_sk)
               ->  Index Only Scan using promotion_pkey on promotion  (cost=0.00..0.40 rows=1 width=4)
                     Index Cond: (p_promo_sk = store_sales.ss_promo_sk)
         ->  Index Scan using customer_address_pkey on customer_address ad2  (cost=0.00..0.68 rows=1 width=368)
               Index Cond: (ca_address_sk = customer.c_current_addr_sk)
(25 rows)
```

9.3.9.5 Scan Operation Hints

Function

These hints specify a scan operation, which can be **tablescan**, **indexscan**, or **indexonlyscan**.

Syntax

```
[no] tablescan|indexscan|indexonlyscan(table [index])
```

Parameter Description

- **no** specifies that the specified hint will not be used for scanning.
- **table** specifies the table to be scanned. You can specify only one table. Use a table alias (if any) instead of a table name.
- **index** specifies the index for **indexscan** or **indexonlyscan**. You can specify only one index.

NOTE

indexscan and **indexonlyscan** hints can be used only when the specified index belongs to the table.

Scan operation hints can be used for row-store tables, column-store tables, HDFS tables, OBS tables, and subquery tables. (The current feature is a lab feature. Contact Huawei technical support before using it.) HDFS internal tables include base tables and delta tables. The delta tables are invisible to users. Therefore, scan operation hints are used only for base tables.

Example

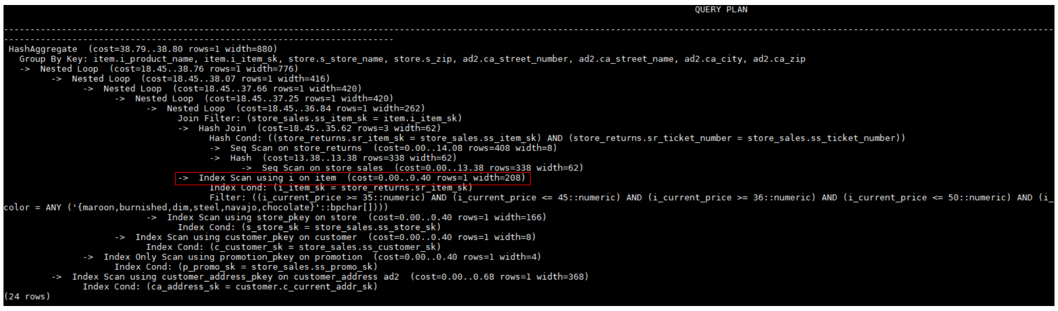
To specify an index-based hint for a scan, create an index named **i** on the **i_item_sk** column of the **item** table.

```
create index i on item(i_item_sk);
```

Hint the query plan in **Examples** as follows:

```
explain
select /*+ indexscan(item i) */ i_product_name product_name ...
```

item is scanned based on an index. The optimized plan is as follows:



```
                                                                    QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------
 HashAggregate  (cost=38.79..38.80 rows=1 width=880)
   Group By Key: item.i_product_name, item.i_item_sk, store.s_store_name, store.s_zip, ad2.ca_street_number, ad2.ca_street_name, ad2.ca_city, ad2.ca_zip
   ->  Nested Loop  (cost=18.45..38.76 rows=1 width=776)
         ->  Nested Loop  (cost=18.45..38.07 rows=1 width=416)
               ->  Nested Loop  (cost=18.45..37.66 rows=1 width=420)
                     ->  Nested Loop  (cost=18.45..37.25 rows=1 width=420)
                           ->  Nested Loop  (cost=18.45..36.84 rows=1 width=262)
                                 Join Filter: (store_sales.ss_item_sk = item.i_item_sk)
                                 ->  Hash Join  (cost=18.45..35.62 rows=3 width=62)
                                       Hash Cond: ((store_returns.sr_item_sk = store_sales.ss_item_sk) AND (store_returns.sr_ticket_number = store_sales.ss_ticket_number))
                                       ->  Seq Scan on store_returns  (cost=0.00..14.08 rows=408 width=8)
                                       ->  Hash  (cost=13.38..13.38 rows=338 width=62)
                                             ->  Seq Scan on store sales  (cost=0.00..13.38 rows=338 width=62)
                                 ->  Index Scan using i on item  (cost=0.00..0.40 rows=1 width=208)
                                       Index Cond: (i_item_sk = store_returns.sr_item_sk)
                                       Filter: ((i_current_price >= 35::numeric) AND (i_current_price <= 45::numeric) AND (i_current_price >= 36::numeric) AND (i_current_price <= 50::numeric) AND (i_
color = ANY ('{maroon,burnished,dim,steel,navajo,chocolate}'::bpchar[])))
                           ->  Index Scan using store_pkey on store  (cost=0.00..0.40 rows=1 width=166)
                                 Index Cond: (s_store_sk = store_sales.ss_store_sk)
                     ->  Index Scan using customer_pkey on customer  (cost=0.00..0.40 rows=1 width=8)
                           Index Cond: (c_customer_sk = store_sales.ss_customer_sk)
               ->  Index Only Scan using promotion_pkey on promotion  (cost=0.00..0.40 rows=1 width=4)
                     Index Cond: (p_promo_sk = store_sales.ss_promo_sk)
         ->  Index Scan using customer_address_pkey on customer_address ad2  (cost=0.00..0.68 rows=1 width=368)
               Index Cond: (ca_address_sk = customer.c_current_addr_sk)
(24 rows)
```

9.3.9.6 Sublink Name Hints

Function

These hints specify the name of a sublink block.

Syntax

```
blockname (table)
```

Parameter Description

- *table* specifies the name you have specified for a sublink block.

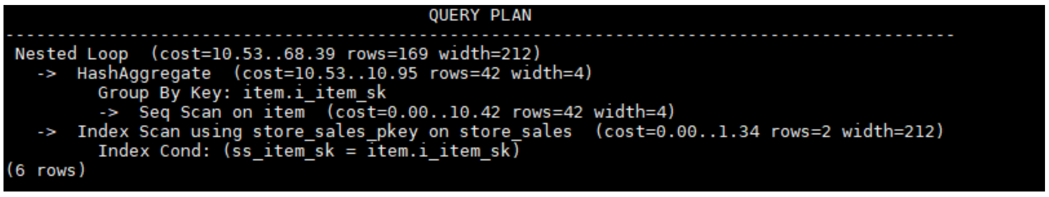
NOTE

- The **blockname** hint is used by an outer query only when the corresponding sublink is not pulled up. Currently, only the **Agg** equivalent join, **IN**, and **EXISTS** sublinks can be pulled up. This hint is usually used together with the hints described in the previous sections.
- The subquery after the **FROM** keyword is hinted by using the subquery alias. In this case, **blockname** becomes invalid.
- If a sublink contains multiple tables, the tables will be joined with the outer-query tables in a random sequence after the sublink is pulled up. In this case, **blockname** also becomes invalid.

Example

```
explain select /*+nestloop(store_sales tt) */ * from store_sales where ss_item_sk in (select /*
+blockname(tt)*/ i_item_sk from item group by 1);
```

tt indicates the sublink block name. After being pulled up, the sublink is joined with the outer-query table **store_sales** by using **nestloop**. The optimized plan is as follows:



```
                                  QUERY PLAN
-----------------------------------------------------------------------------------------
 Nested Loop  (cost=10.53..68.39 rows=169 width=212)
   ->  HashAggregate  (cost=10.53..10.95 rows=42 width=4)
         Group By Key: item.i_item_sk
         ->  Seq Scan on item  (cost=0.00..10.42 rows=42 width=4)
   ->  Index Scan using store_sales_pkey on store_sales  (cost=0.00..1.34 rows=2 width=212)
         Index Cond: (ss_item_sk = item.i_item_sk)
(6 rows)
```

9.3.9.7 Hint Errors, Conflicts, and Other Warnings

Plan hints change an execution plan. You can run **EXPLAIN** to view the changes.

Hints containing errors are invalid and do not affect statement execution. The errors will be displayed in different ways based on statement types. Hint errors in an **EXPLAIN** statement are displayed as a warning on the interface. Hint errors in other statements will be recorded in debug1-level logs containing the **PLANHINT** keyword.

Hint error types are as follows:

- Syntax errors

  An error will be reported if the syntax tree fails to be reduced. The No. of the row generating an error is displayed in the error details.

  For example, the hint keyword is incorrect, no table or only one table is specified in the **leading** or **join** hint, or no tables are specified in other hints. The parsing of a hint is terminated immediately after a syntax error is detected. Only the hints that have been parsed successfully are valid.

  For example:

  ```
  leading((t1 t2)) nestloop(t1) rows(t1 t2 #10)
  ```

  The syntax of **nestloop(t1)** is wrong and its parsing is terminated. Only **leading(t1 t2)** that has been successfully parsed before **nestloop(t1)** is valid.

- Semantic errors
  - An error will be reported if the specified tables do not exist, multiple tables are found based on the hint setting, or a table is used more than once in the **leading** or **join** hint.
  - An error will be reported if the index specified in a scan hint does not exist.
  - If multiple tables with the same name exist after a subquery is pulled up and some of them need to be hinted, add aliases for them to avoid name duplication.

- Duplicated or conflicted hints

  If hint duplication or conflicts occur, only the first hint takes effect. A message will be displayed to describe the situation.

  - Hint duplication indicates that a hint is used more than once in the same query, for example, **nestloop(t1 t2) nestloop(t1 t2)**.
  - A hint conflict indicates that the functions of two hints with the same table list conflict with each other.

    For example, if **nestloop (t1 t2) hashjoin (t1 t2)** is used, **hashjoin (t1 t2)** becomes invalid. **nestloop(t1 t2)** does not conflict with **no mergejoin(t1 t2)**.

    NOTICE

    The table list in the **leading** hint is disassembled. For example, **leading ((t1 t2 t3))** will be disassembled as **leading((t1 t2)) leading(((t1 t2) t3))**, which will conflict with **leading((t2 t1))** (if any). In this case, the latter **leading(t2 t1)** becomes invalid. If two hints use duplicated table lists and only one of them has the specified outer/inner table, the one without a specified outer/inner table becomes invalid.

- A hint becomes invalid after a sublink is pulled up.

  In this case, a message will be displayed. Generally, such invalidation occurs when a sublink contains multiple tables to be joined. After the sublink is pulled up, the tables will not be join members.
- Hints are not used.
  - If a **hashjoin** or **mergejoin** hint is specified for non-equivalent joins, it will not be used.
  - If an **indexscan** or **indexonlyscan** hint is specified for a table that does not have an index, it will not be used.
  - If an **indexscan** or **indexonlyscan** hint is specified for a full-table scan, it will not be used. Generally, index paths are generated only when filtering conditions are used on index columns. Indexes are not used during a full table scan.
  - If an **indexonlyscan** hint is specified when the output or predicate condition column does not contain only indexes, it will not be used.
  - In equivalent joins, only the joins containing equivalence conditions are valid. Therefore, the **leading**, **join**, and **rows** hints specified for the joins without an equivalence condition will not be used. For example, **t1**, **t2**, and **t3** are to be joined, and the join between **t1** and **t3** does not contain an equivalence condition. In this case, **leading(t1 t3)** will not be used.
  - If no sublink is pulled up, the specified **blockname** hint will not be used.
  - Skew hints are not used possibly because:
    - Skew information specified in hints is incorrect or incomplete, for example, no value is specified for join optimization.
    - Skew optimization is disabled by GUC parameters.

9.3.9.8 Optimizer GUC Parameter Hints

Function

Sets GUC parameters related to query optimization that take effect during the query execution. For details about the application scenarios of hints, see the description of each GUC parameter.

Syntax

```
set(param value)
```

Parameters

- **param** indicates the parameter name.
- **value** indicates the value of a parameter.
- Currently, the following parameters can be set and take effect by using Hint:
  - Boolean

    enable_bitmapscan, **enable_hashagg**, **enable_hashjoin**, **enable_indexscan**, **enable_indexonlyscan**, **enable_material**, **enable_mergejoin**, **enable_nestloop**, **enable_index_nestloop**, **enable_seqscan**, **enable_sort**, and **enable_tidscan**

- Integer

  query_dop
- Floating point

  cost_weight_index, **default_limit_rows**, **seq_page_cost**, **random_page_cost**, **cpu_tuple_cost**, **cpu_index_tuple_cost**, **cpu_operator_cost**, and **effective_cache_size**
- Enumeration

  try_vector_engine_strategy

NOTE

- If you set a parameter that is not in the whitelist and the parameter value is invalid or the hint syntax is incorrect, the query execution is not affected. Run **explain(verbose on)**. An error message is displayed, indicating that hint parsing fails.
- The GUC parameter hint takes effect only in the outermost query. That is, the GUC parameter hint in the subquery does not take effect.
- The GUC parameter hint in the view definition does not take effect.
- In the **CREATE TABLE ... AS ...** statement, the outermost GUC parameter hint takes effect.

9.3.9.9 Hint for Selecting the Custom Plan or Generic Plan

Function

For query statements and DML statements executed in PBE mode, the optimizer generates a custom plan or generic plan based on factors such as rules, costs, and parameters. You can use the hint of **use_cplan** or **use_gplan** to specify the plan to execute.

Syntax

- To select the custom plan, run the following statement:
```
use_cplan
```
- To select the generic plan, run the following statement:
```
use_gplan
```

NOTE

- For SQL statements that are executed in non-PBE mode, setting this hint does not affect the execution mode.
- This hint has a higher priority than cost-based selection and the **plan_cache_mode** parameter. That is, this hint does not take effect for statements for which **plan_cache_mode** cannot be forcibly set to specify an execution mode.

Examples

Forcibly use the custom plan.

```
create table t (a int, b int, c int);
prepare p as select /*+ use_cplan */ * from t where a = $1;
explain execute p(1);
```

In the following plan, the filtering condition is the actual value of the input parameter, that is, the plan is a custom plan.

```
                          QUERY PLAN
----------------------------------------------------
 Seq Scan on t  (cost=0.00..34.31 rows=10 width=12)
   Filter: (a = 1)
(2 rows)
```

Forcibly use the generic plan.

```
deallocate p;
prepare p as select /*+ use_gplan */ * from t where a = $1;
explain execute p(1);
```

In the following plan, the filtering condition is the input parameter to be added. That is, the plan is a generic plan.

```
                          QUERY PLAN
----------------------------------------------------
 Seq Scan on t  (cost=0.00..34.31 rows=10 width=12)
   Filter: (a = $1)
(2 rows)
```

9.3.9.10 Hint Specifying Not to Expand Subqueries

Function

When the database optimizes the query logic, some subqueries can be promoted to the upper layer to avoid nested execution. However, for some subqueries that have a low selection rate and can use indexes to filter access pages, nested execution does not cause too much performance deterioration, while after the promotion, the query search scope is expanded, which may cause performance deterioration. In this case, you can use the **no_expand** hint for debugging. This hint is not recommended in most cases.

Syntax

```
no_expand
```

Examples

Normal query execution:

```
explain select * from t1 where t1.a in (select t2.a from t2);
```

Plan:

```
                              QUERY PLAN
-----------------------------------------------------------------------
 Hash Join  (cost=38.81..92.58 rows=972 width=12)
   Hash Cond: (t1.a = t2.a)
   ->  Seq Scan on t1  (cost=0.00..29.45 rows=1945 width=12)
   ->  Hash  (cost=36.31..36.31 rows=200 width=4)
         ->  HashAggregate  (cost=34.31..36.31 rows=200 width=4)
               Group By Key: t2.a
               ->  Seq Scan on t2  (cost=0.00..29.45 rows=1945 width=4)
(7 rows)
```

After **no_expand** is added:

```
explain select * from t1 where t1.a in (select /*+ no_expand*/ t2.a from t2);
```

Plan:

```
                              QUERY PLAN
--------------------------------------------------------------
 Seq Scan on t1  (cost=34.31..68.62 rows=972 width=12)
   Filter: (hashed SubPlan 1)
   SubPlan 1
     ->  Seq Scan on t2  (cost=0.00..29.45 rows=1945 width=4)
(4 rows)
```

9.3.9.11 Hint Specifying Not to Use Global Plan Cache

Function

When global plan cache is enabled, you can use the **no_gpc** hint to force a single query statement not to share the plan cache globally. Only the plan cache within the current session lifecycle is retained.

The current feature is a lab feature. Contact Huawei technical support before using it.

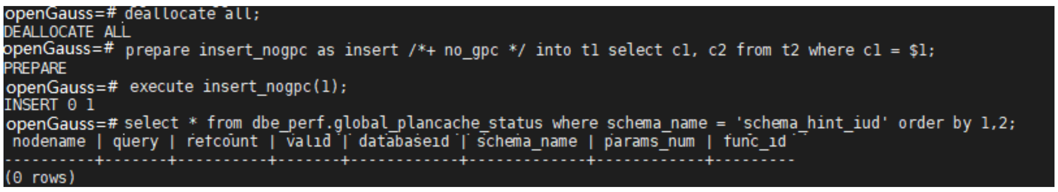
Syntax

```
no_gpc
```

NOTE

This parameter takes effect only for statements executed by PBE when **enable_global_plancache** is set to **on**.

Example



```
openGauss=# deallocate all;
DEALLOCATE ALL
openGauss=# prepare insert_nogpc as insert /*+ no_gpc */ into t1 select c1, c2 from t2 where c1 = $1;
PREPARE
openGauss=# execute insert_nogpc(1);
INSERT 0 1
openGauss=# select * from dbe_perf.global_plancache_status where schema_name = 'schema_hint_iud' order by 1,2;
 nodename | query | refcount | valid | databaseid | schema_name | params_num | func_id
----------+-------+----------+-------+------------+-------------+------------+---------
(0 rows)
```

No result exists in the **dbe_perf.global_plancache_status** view, that is, no plan is cached globally.

9.3.9.12 Hint of Parameterized Paths at the Same Level

Function

The **predpush_same_level** hint is used to specify the generation of parameterized paths between tables or MVs at the same level.

Syntax

```
predpush_same_level(src, dest)
predpush_same_level(src1 src2 ..., dest)
```

NOTE

This parameter takes effect only when the **predpushforce** option in **rewrite_rule** is enabled.

Examples

Prepare parameters, tables, and indexes.

```
openGauss=# set rewrite_rule = 'predpushforce';
SET
openGauss=# create table t1(a int, b int);
CREATE TABLE
openGauss=# create table t2(a int, b int);
CREATE TABLE
openGauss=# create index idx1 on t1(a);
CREATE INDEX
openGauss=# create index idx2 on t2(a);
CREATE INDEX
```

Run the following statement to view the plan:

```
openGauss=# explain select * from t1, t2 where t1.a = t2.a;
                          QUERY PLAN
------------------------------------------------------------------
 Hash Join  (cost=27.50..56.25 rows=1000 width=16)
   Hash Cond: (t1.a = t2.a)
   ->  Seq Scan on t1  (cost=0.00..15.00 rows=1000 width=8)
   ->  Hash  (cost=15.00..15.00 rows=1000 width=8)
         ->  Seq Scan on t2  (cost=0.00..15.00 rows=1000 width=8)
(5 rows)
```

The filter condition **t1.a = t2.a** is displayed on **Join**. In this case, **predpush_same_level(t1, t2)** can be used to push the condition down to the scan operator of t2.

```
openGauss=# explain select /*+predpush_same_level(t1, t2)*/ * from t1, t2 where t1.a = t2.a;
                           QUERY PLAN
---------------------------------------------------------------------
 Nested Loop  (cost=0.00..335.00 rows=1000 width=16)
   ->  Seq Scan on t1  (cost=0.00..15.00 rows=1000 width=8)
   ->  Index Scan using idx2 on t2  (cost=0.00..0.31 rows=1 width=8)
         Index Cond: (a = t1.a)
(4 rows)
```

NOTICE

- **predpush_same_level** can specify multiple **src** parameters in the same condition.
- If the specified **src** and **dest** conditions do not exist or do not meet the parameterized path requirements, this hint does not take effect.

9.3.9.13 Hint for Materializing a Sub-plan Result

Function

You can materialize a sub-plan result to temporarily store the query record. This hint is used only in INSERT statements.

Syntax

```
material_subplan
```

Examples

Normal INSERT INTO...SELECT statement:

```
insert into test select /*+ nestloop(test_src t1)*/ * from test_src where notexists(select 1 from test t1 where
t1.a = test_src.a);
```

Execution plan:

```
                     QUERY PLAN
------------------------------------------------------------
 Insert on test
   ->  Nested Loop Anti Join
         ->  Seq Scan on test_src
         ->  Index Only Scan using test_a_idx on test t1
               Index Cond: (a = test_src.a)
(5 rows)
```

Use the material_subplan hint operator:

```
insert /*+ material_subplan*/ into test select /*+ nestloop(test_src t1)*/ * from test_src where not
exists(select 1 from test t1 where t1.a = test_src.a);
```

The execution plan is as follows:

```
                        QUERY PLAN
------------------------------------------------------------------
 Insert on test
   ->  Materialize
         ->  Nested Loop Anti Join
               ->  Seq Scan on test_src
               ->  Index Only Scan using test_a_idx on test t1
                     Index Cond: (a = test_src.a)
(6 rows)
```

9.3.10 Using the Vectorized Executor for Tuning

GaussDB supports the row executor and vectorized executor for processing row-store tables and column-store tables, respectively.

- More data is read in one batch at a time, saving I/O resources.
- There are a large number of records in a batch, and the CPU cache hit rate increases.
- In pipeline mode, the number of function calls is small.
- A batch of data is processed at a time, which is efficient.

Therefore, GaussDB can achieve better query performance for complex analytical queries. However, column-store tables do not perform well in data insertion and update. Therefore, column-store tables cannot be used for services with frequent data insertion and update.

To improve the query performance of row-store tables in complex analytical queries, GaussDB provides the vectorized executor for processing row-store tables. You can set **try_vector_engine_strategy** to convert query statements containing row-store tables into vectorized execution plans for execution.

This conversion is not applicable to all query scenarios. If a query statement contains operations such as expression calculation, multi-table join, and aggregation, the performance can be improved by converting the statement to a vectorized execution plan. Theoretically, converting a row-store table to a vectorized execution plan causes conversion overheads and performance deterioration. After the foregoing expression calculation, join operation, and aggregation operations are converted into vectorized execution plans, performance can be improved. The performance improvement must be higher than the overheads generated by the conversion. This determines whether the conversion is required.

Take TPCH Q1 as an example. When the row executor is used, the execution time of the scan operator is 405210 ms, and the execution time of the aggregation operation is 2618964 ms. After the vectorized executor is used, the execution time of the scan operator (SeqScan and VectorAdapter) is 470840 ms, and the execution time of the aggregation operation is 212384 ms. So the query performance can be improved.

Execution plan of the TPCH Q1 row executor:

```
                                                QUERY PLAN
--------------------------------------------------------------------------------------------------------------------------------
-------------
 Sort  (cost=43539570.49..43539570.50 rows=6 width=260) (actual time=3024174.439..3024174.439 rows=4 
loops=1)
   Sort Key: l_returnflag, l_linestatus
   Sort Method: quicksort  Memory: 25kB
   ->  HashAggregate  (cost=43539570.30..43539570.41 rows=6 width=260) (actual 
time=3024174.396..3024174.403 rows=4 loops=1)
         Group By Key: l_returnflag, l_linestatus
         ->  Seq Scan on lineitem  (cost=0.00..19904554.46 rows=590875396 width=28) (actual 
time=0.016..405210.038 rows=596140342 loops=1)
               Filter: (l_shipdate <= '1998-10-01 00:00:00'::timestamp without time zone)
               Rows Removed by Filter: 3897560
 Total runtime: 3024174.578 ms
(9 rows)
```

Execution plan of the TPCH Q1 vectorized executor:

```
                                                          QUERY 
PLAN
--------------------------------------------------------------------------------------------------------------------------------
-------------------------------------
 Row Adapter  (cost=43825808.18..43825808.18 rows=6 width=298) (actual time=683224.925..683224.927 
rows=4 loops=1)
   ->  Vector Sort  (cost=43825808.16..43825808.18 rows=6 width=298) (actual 
time=683224.919..683224.919 rows=4 loops=1)
         Sort Key: l_returnflag, l_linestatus
         Sort Method: quicksort  Memory: 3kB
         ->  Vector Sonic Hash Aggregate  (cost=43825807.98..43825808.08 rows=6 width=298) (actual 
time=683224.837..683224.837 rows=4 loops=1)
               Group By Key: l_returnflag, l_linestatus
               ->  Vector Adapter(type: BATCH MODE)  (cost=19966853.54..19966853.54 rows=596473861 
width=66) (actual time=0.982..470840.274 rows=596140342 loops=1)
                     Filter: (l_shipdate <= '1998-10-01 00:00:00'::timestamp without time zone)
                     Rows Removed by Filter: 3897560
                     ->  Seq Scan on lineitem  (cost=0.00..19966853.54 rows=596473861 width=66) (actual 
time=0.364..199301.737 rows=600037902 loops=1)
 Total runtime: 683225.564 ms
(11 rows)
```

9.4 Optimization Cases

9.4.1 Case: Modifying the GUC Parameter rewrite_rule

rewrite_rule contains multiple query rewriting rules: **magicset**, **partialpush**, **uniquecheck**, **disablerep**, **intargetlist**, and **predpush**. The following describes the application scenarios of some important rules:

Promoting the Subquery in the Target Column Using intargetlist

The query performance can be greatly improved by converting the subquery in the target column to JOIN. The following is an example:

```
openGauss=#  set rewrite_rule='none';
SET
openGauss=# create table t1(c1 int,c2 int);
CREATE TABLE
openGauss=# create table t2(c1 int,c2 int);
CREATE TABLE
openGauss=#  explain (verbose on, costs off) select c1,(select avg(c2) from t2 where t2.c2=t1.c2) from t1
where t1.c1<100 order by t1.c2;
              QUERY PLAN
-----------------------------------------------
 Sort
   Output: t1.c1, ((SubPlan 1)), t1.c2
   Sort Key: t1.c2
   ->  Seq Scan on public.t1
         Output: t1.c1, (SubPlan 1), t1.c2
         Filter: (t1.c1 < 100)
         SubPlan 1
           ->  Aggregate
                 Output: avg(t2.c2)
                 ->  Seq Scan on public.t2
                       Output: t2.c1, t2.c2
                       Filter: (t2.c2 = t1.c2)
(12 rows)
```

Because the subquery **(select avg(c2) from t2 where t2.c2=t1.c2)** in the target column cannot be pulled up, execution of the subquery is triggered each time a row of data of **t1** is scanned, and the query efficiency is low. If the **intargetlist** parameter is enabled, the subquery is converted to JOIN to improve the query performance.

```
openGauss=#  set rewrite_rule='intargetlist';
SET
openGauss=# explain (verbose on, costs off) select c1,(select avg(c2) from t2 where t2.c2=t1.c2) from t1
where t1.c1<100 order by t1.c2;
              QUERY PLAN
-----------------------------------------------
 Sort
   Output: t1.c1, (avg(t2.c2)), t1.c2
   Sort Key: t1.c2
   ->  Hash Left Join
         Output: t1.c1, (avg(t2.c2)), t1.c2
         Hash Cond: (t1.c2 = t2.c2)
         ->  Seq Scan on public.t1
               Output: t1.c1, t1.c2
               Filter: (t1.c1 < 100)
         ->  Hash
               Output: (avg(t2.c2)), t2.c2
               ->  HashAggregate
                     Output: avg(t2.c2), t2.c2
                     Group By Key: t2.c2
                     ->  Seq Scan on public.t2
                           Output: t2.c2
(16 rows)
```

Promoting the Subquery Without Aggregate Using uniquecheck

Ensure that each condition has only one line of output. The subqueries with aggregate functions can be automatically pulled up. For subqueries without aggregate functions, the following is an example:

select t1.c1 from t1 where t1.c1 = (select t2.c1 from t2 where t1.c1=t2.c2) ;

Rewrite as follows:

select t1.c1 from t1 join (select t2.c1 from t2 where t2.c1 is not null group by t2.c1(unique check)) tt(c1) on tt.c1=t1.c1;

To ensure semantic equivalence, the subquery **tt** must ensure that each **group by t2.c1** has only one line of output. Enable the **uniquecheck** query rewriting parameter to ensure that the query can be pulled up and equivalent. If more than one row of data is output at run time, an error is reported.

```
openGauss=# set rewrite_rule='uniquecheck';
SET
openGauss=#  explain verbose select t1.c1 from t1 where t1.c1 = (select t2.c1 from t2 where t1.c1=t2.c1);
                                 QUERY PLAN
------------------------------------------------------------------------------------
 Hash Join  (cost=43.36..104.40 rows=2149 distinct=[200, 200] width=4)
   Output: t1.c1
   Hash Cond: (t1.c1 = subquery."?column?")
   ->  Seq Scan on public.t1  (cost=0.00..31.49 rows=2149 width=4)
         Output: t1.c1, t1.c2
   ->  Hash  (cost=40.86..40.86 rows=200 width=8)
         Output: subquery."?column?", subquery.c1
         ->  Subquery Scan on subquery  (cost=36.86..40.86 rows=200 width=8)
               Output: subquery."?column?", subquery.c1
               ->  HashAggregate  (cost=36.86..38.86 rows=200 width=4)
                     Output: t2.c1, t2.c1
                     Group By Key: t2.c1
                     Filter: (t2.c1 IS NOT NULL)
                     Unique Check Required
                     ->  Seq Scan on public.t2  (cost=0.00..31.49 rows=2149 width=4)
                           Output: t2.c1
(16 rows)
```

Note: Because **group by t2.c1 unique check** occurs before the filter condition **tt.c1=t1.c1**, an error may be reported after the query that does not report an error is rewritten. An example is as follows:

There are tables **t1** and **t2**. The data in the tables is as follows:

```
openGauss=#  select * from t1 order by c2;
 c1 | c2
----+----
  1 |  1
  2 |  2
  3 |  3
(3 rows)
openGauss=#  select * from t2 order by c2;
 c1 | c2
----+----
  1 |  1
  2 |  2
  3 |  3
  4 |  4
  4 |  4
  5 |  5
(6 rows)
```

Disable and enable the **uniquecheck** parameter for comparison. After the parameter is enabled, an error is reported.

```
openGauss=#  select t1.c1 from t1 where t1.c1 = (select t2.c1 from t2 where t1.c1=t2.c2) ;
 c1
----
  1
  2
  3
(3 rows)
```

```
openGauss=#  set rewrite_rule='uniquecheck';
SET
openGauss=#  select t1.c1 from t1 where t1.c1 = (select t2.c1 from t2 where t1.c1=t2.c2) ;
ERROR:  more than one row returned by a subquery used as an expression
```

9.4.2 Case: Adjusting I/O Parameters to Reduce the Log Bloat Rate

- Parameter values before adjustment:
  - pagewriter_sleep=2000ms
  - bgwriter_delay=2000ms
  - max_io_capacity=500MB
- Parameter values after adjustment:
  - pagewriter_sleep=100ms
  - bgwriter_delay=1s
  - max_io_capacity=300MB

NOTE

- The **max_io_capacity** parameter is set to a small value because the I/O does not use the maximum value of the previous parameter. This parameter is used to limit the upper limit of the I/O usage of the backend write process.
- Log recycling is triggered only when the number of logs reaches a certain value. The formula for calculating the value is as follows: Value of **wal_keep_segments** + Value of **checkpoint_segments** x 2 + 1. If **checkpoint_segments** is set to **128** and **wal_keep_segments** is set to **128**, the number of logs is (128 + 128 x 2 + 1) x 16 MB = 6 GB.
- Before the parameters are adjusted, the Xlogs of different data volumes bloat in different degrees in the TPC-C data export phase. As a result, GB-level logs bloat. The main cause is that dirty pages are not flushed to disks, the recovery point cannot be pushed forward, and logs cannot be recycled in time. After the parameters are adjusted, the log bloat rate decreases significantly.
- Take the data warehouse 2000 as an example. Before the parameter adjustment, the log size bloats by 10 GB in the data export phase. After the parameter adjustment, the log size remains within the range of the minimum xlog value calculated based on the parameter setting.

9.4.3 Case: Creating an Appropriate Index

Symptom

Query the information about all personnel in the sales department.

```
SELECT staff_id,first_name,last_name,employment_id,state_name,city
FROM staffs,sections,states,places
WHERE sections.section_name='Sales'
AND staffs.section_id = sections.section_id
AND sections.place_id = places.place_id
AND places.state_id = states.state_id
ORDER BY staff_id;
```

Optimization Analysis

The original execution plan is as follows before creating the **places.place_id** and **states.state_id** indexes:

```
                                             QUERY PLAN
-----------------------------------------------------------------------------------------------------
 Sort  (cost=129.74..131.18 rows=576 width=136)
   Sort Key: staffs.staff_id
   ->  Hash Join  (cost=70.54..103.33 rows=576 width=136)
         Hash Cond: (states.state_id = places.state_id)
         ->  Seq Scan on states  (cost=0.00..22.38 rows=1238 width=36)
         ->  Hash  (cost=69.38..69.38 rows=93 width=108)
               ->  Hash Join  (cost=42.41..69.38 rows=93 width=108)
                     Hash Cond: (places.place_id = sections.place_id)
                     ->  Seq Scan on places  (cost=0.00..21.67 rows=1167 width=40)
                     ->  Hash  (cost=42.21..42.21 rows=16 width=76)
                           ->  Hash Join  (cost=24.66..42.21 rows=16 width=76)
                                 Hash Cond: (staffs.section_id = sections.section_id)
                                 ->  Seq Scan on staffs  (cost=0.00..15.37 rows=537 width=76)
                                 ->  Hash  (cost=24.59..24.59 rows=6 width=8)
                                       ->  Seq Scan on sections  (cost=0.00..24.59 rows=6 width=8)
                                             Filter: (section_name = 'Sales'::text)
(16 rows)
```

The optimized execution plan is as follows (two indexes have been created on the **places.place_id** and **states.state_id** columns):

```
                                                 QUERY PLAN
-------------------------------------------------------------------------------------------------------------
 Sort  (cost=119.76..121.20 rows=576 width=136)
   Sort Key: staffs.staff_id
   ->  Hash Join  (cost=70.14..93.35 rows=576 width=136)
         Hash Cond: (staffs.section_id = sections.section_id)
         ->  Seq Scan on staffs  (cost=0.00..15.37 rows=537 width=76)
         ->  Hash  (cost=67.43..67.43 rows=217 width=68)
               ->  Nested Loop  (cost=24.66..67.43 rows=217 width=68)
                     ->  Hash Join  (cost=24.66..51.06 rows=35 width=40)
                           Hash Cond: (places.place_id = sections.place_id)
                           ->  Seq Scan on places  (cost=0.00..21.67 rows=1167 width=40)
                           ->  Hash  (cost=24.59..24.59 rows=6 width=8)
                                 ->  Seq Scan on sections  (cost=0.00..24.59 rows=6 width=8)
                                       Filter: (section_name = 'Sales'::text)
                     ->  Index Scan using states_state_id_idx on states  (cost=0.00..0.41 rows=6 width=36)
                           Index Cond: (state_id = places.state_id)
(15 rows)
```

9.4.4 Case: Adding NOT NULL for JOIN Columns

```
SELECT * FROM join_a a JOIN join_b b ON a.b = b.b;
```

The execution plan is as follows:

```
                                                      QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Hash Join  (cost=58.35..14677.69 rows=1074607 width=16) (actual time=23.374..23.384 rows=10 loops=1)
   Hash Cond: (a.b = b.b)
   ->  Seq Scan on join_a a  (cost=0.00..2248.10 rows=100010 width=8) (actual time=0.495..12.551
rows=100010 loops=1)
   ->  Hash  (cost=31.49..31.49 rows=2149 width=8) (actual time=0.614..0.614 rows=1000 loops=1)
          Buckets: 32768  Batches: 1  Memory Usage: 40kB
         ->  Seq Scan on join_b b  (cost=0.00..31.49 rows=2149 width=8) (actual time=0.009..0.183 rows=1000
loops=1)
 Total runtime: 23.716 ms
(7 rows)
```

Optimization Analysis

1. According to the execution plan, the sequential scan phase is time consuming.
2. Therefore, you are advised to manually add **NOT NULL** for the **JOIN** column in the statement, as shown below:

```
SELECT
 *
SELECT * FROM join_a a JOIN join_b b ON a.b = b.b where a.b IS NOT NULL;
```

The execution plan is as follows:

```
                                                         QUERY PLAN
---------------------------------------------------------------------------------------------------------------------
 Hash Join  (cost=58.22..14560.97 rows=1063762 width=16) (actual time=13.237..13.247 rows=10
loops=1)
   Hash Cond: (a.b = b.b)
   ->  Seq Scan on join_a a  (cost=0.00..2248.10 rows=99510 width=8) (actual time=12.417..12.422
rows=10 loops=1)
         Filter: (b IS NOT NULL)
         Rows Removed by Filter: 100000
   ->  Hash  (cost=31.49..31.49 rows=2138 width=8) (actual time=0.566..0.566 rows=1000 loops=1)
          Buckets: 32768  Batches: 1  Memory Usage: 40kB
         ->  Seq Scan on join_b b  (cost=0.00..31.49 rows=2138 width=8) (actual time=0.011..0.229
rows=1000 loops=1)
               Filter: (b IS NOT NULL)
 Total runtime: 13.556 ms
(10 rows)
```

9.4.5 Case: Modifying a Partitioned Table

Symptom

In the following simple SQL statements, the performance bottlenecks exist in the scan operation on the **normal_date** table.

```
                                                         QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------
-----------------
 Seq Scan on normal_date  (cost=0.00..259.00 rows=30 width=12) (actual time=0.100..3.466 rows=30
loops=1)
   Filter: (("time" >= '2022-09-01 00:00:00'::timestamp without time zone) AND ("time" <= '2022-10-01
00:00:00'::timestamp without time zone))
   Rows Removed by Filter: 9970
 Total runtime: 3.587 ms
(4 rows)
```

Optimization Analysis

Obviously, there are date features in the **time** column of table data in the service layer, and this meet the features of a partitioned table. Replan the table definition of the **normal_date** table. Set the **time** column as a partition key, and month as an interval unit. Define the partitioned table **normal_date_part**. The modified result is as follows, and the performance is improved by nearly 10 times.

```
                                                         QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------
-----------------------
 Partition Iterator  (cost=0.00..480.00 rows=30 width=12) (actual time=0.038..0.085 rows=30 loops=1)
   Iterations: 2
   ->  Partitioned Seq Scan on normal_date_part  (cost=0.00..480.00 rows=30 width=12) (actual
time=0.049..0.063 rows=30 loops=2)
         Filter: (("time" >= '2022-09-01 00:00:00'::timestamp without time zone) AND ("time" <= '2022-10-01
00:00:00'::timestamp without time zone))
         Rows Removed by Filter: 31
         Selected Partitions:  3..4
 Total runtime: 0.360 ms
(7 rows)
```

9.4.6 Case: Rewriting SQL and Deleting Subqueries (1)

Symptom

```
select
    1,
    (select count(*) from normal_date n where n.id = a.id) as GZCS
from normal_date a;
```

This SQL performance is poor. SubPlan exists in the execution plan as follows:

```
                                                    QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------
---------
 Seq Scan on normal_date a  (cost=0.00..888118.42 rows=5129 width=4) (actual time=2.394..22194.907
rows=10000 loops=1)
   SubPlan 1
     ->  Aggregate  (cost=173.12..173.12 rows=1 width=8) (actual time=22179.496..22179.942 rows=10000
loops=10000)
           ->  Seq Scan on normal_date n  (cost=0.00..173.11 rows=1 width=0) (actual
time=11279.349..22159.608 rows=10000 loops=10000)
                 Filter: (id = a.id)
                 Rows Removed by Filter: 99990000
 Total runtime: 22196.415 ms
(7 rows)
```

Optimization

The core of this optimization is to eliminate subqueries. Based on the service scenario analysis, *a.id* is not null. In terms of SQL syntax, you can rewrite the SQL statement as follows:

```
select
count(*)
from normal_date n, normal_date a
where n.id = a.id
group by  a.id;
The plan is as follows:
                                                   QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------
----
 HashAggregate  (cost=480.86..532.15 rows=5129 width=12) (actual time=21.539..24.356 rows=10000
loops=1)
   Group By Key: a.id
   ->  Hash Join  (cost=224.40..455.22 rows=5129 width=4) (actual time=6.402..13.484 rows=10000 loops=1)
         Hash Cond: (n.id = a.id)
         ->  Seq Scan on normal_date n  (cost=0.00..160.29 rows=5129 width=4) (actual time=0.087..1.459
rows=10000 loops=1)
         ->  Hash  (cost=160.29..160.29 rows=5129 width=4) (actual time=6.065..6.065 rows=10000 loops=1)
               Buckets: 32768  Batches: 1  Memory Usage: 352kB
               ->  Seq Scan on normal_date a  (cost=0.00..160.29 rows=5129 width=4) (actual time=0.046..2.738
rows=10000 loops=1)
 Total runtime: 26.844 ms
(9 rows)
```

NOTE

To ensure that the modified statements have the same functions, **NOT NULL** is added to *normal_date.id*.

9.4.7 Case: Rewriting SQL and Deleting Subqueries (2)

Symptom

Take the following SQL statement as an example:

```
UPDATE normal_date n SET time = (
    SELECT time FROM normal_date_part p WHERE p.id = n.id
)
WHERE EXISTS
    (SELECT 1
    FROM normal_date_part n2
    WHERE n2.id = n.id);
```

The plan is:

```
                                                                 QUERY PLAN


-----------------------------------------------------------------------------------------------------------------------------------------------------------
 Update on normal_date n  (cost=224.40..2334150.22 rows=5129 width=16) (actual time=17.336..42944.734
rows=10000 loops=1)
   ->  Hash Semi Join  (cost=224.40..2334150.22 rows=5129 width=16) (actual time=16.997..42852.967
rows=10000 loops=1)
         Hash Cond: (n.id = n2.id)
         ->  Seq Scan on normal_date n  (cost=0.00..160.29 rows=5129 width=10) (actual time=0.113..7.271
rows=10000 loops=1)
         ->  Hash  (cost=160.29..160.29 rows=5129 width=10) (actual time=7.381..7.381 rows=10000 loops=1)
               Buckets: 32768  Batches: 1  Memory Usage: 430kB
              ->  Seq Scan on normal_date n2  (cost=0.00..160.29 rows=5129 width=10) (actual
time=0.052..3.501 rows=10000 loops=1)
         SubPlan 1
           ->  Partition Iterator  (cost=0.00..455.00 rows=1 width=8) (actual time=21006.481..42756.884
rows=10000 loops=10000)
                 Iterations: 331
                 ->  Partitioned Seq Scan on normal_date_part p  (cost=0.00..455.00 rows=1 width=8) (actual
time=27228.532..27261.944 rows=10000 loops=3310000)
                       Filter: (id = n.id)
                       Rows Removed by Filter: 99990000
                       Selected Partitions:  1..331
 Total runtime: 42947.153 ms
(15 rows)
```

Optimization

SubPlan exists in the execution plan, and the calculation accounts for a large proportion in the SubPlan query. That is, SubPlan is a performance bottleneck.

Based on the SQL syntax, you can rewrite the SQL statements and delete SubPlan as follows:

```
update normal_date n set time = (
    select time from normal_date_part p where p.id = n.id
);
```

9.4.8 Case: Rewriting SQL Statements and Deleting in-clause

Symptom

in-clause/any-clause is a common SQL statement constraint. Sometimes, the clause following **in** or **any** is a constant. For example:

```
select
count(1)
from calc_empfyc_c1_result_tmp_t1
where ls_pid_cusr1 in ('20120405', '20130405')
```

Or

```
select
count(1)
```

```
from calc_empfyc_c1_result_tmp_t1
where ls_pid_cusr1 in any('20120405', '20130405');
```

Sometimes, the **in** or **any** clause is used as follows:

```
SELECT
*
FROM test1 t1, test2 t2
WHERE t1.a = any(values(t2.a),(t2.b));
```

a and **b** are two columns in **t2**, and **"t1.a = any(values(t2.ba,(t2.b))"** is equivalent to **"t1.a = t2.a or t1.a = t2.b"**.

Therefore, join-condition is essentially an inequality, and nestloop must be used for this join operation. The execution plan is as follows:

```
                                                            QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------
---
 Nested Loop  (cost=0.00..138614.38 rows=2309100 width=16) (actual time=0.152..19225.483 rows=1000
loops=1)
   Join Filter: (SubPlan 1)
   Rows Removed by Join Filter: 999000
   ->  Seq Scan on test1 t1  (cost=0.00..31.49 rows=2149 width=8) (actual time=0.021..3.309 rows=1000
loops=1)
   ->  Materialize  (cost=0.00..42.23 rows=2149 width=8) (actual time=0.331..1265.810 rows=1000000
loops=1000)
         ->  Seq Scan on test2 t2  (cost=0.00..31.49 rows=2149 width=8) (actual time=0.013..0.268 rows=1000
loops=1)
   SubPlan 1
     ->  Values Scan on "*VALUES*"  (cost=0.00..0.03 rows=2 width=4) (actual time=2890.741..7372.739
rows=1999000 loops=1000000)
 Total runtime: 19227.328 ms
(9 rows)
```

Optimization

The test result shows that both result sets are too large. As a result, nestloop is time-consuming with more than one hour to return results. Therefore, the key to performance optimization is to eliminate nestloop, using more efficient hash join. From the perspective of semantic equivalence, the SQL statements can be written as follows:

```
SELECT
*
FROM (
    SELECT * FROM test1 t1, test2 t2 WHERE t1.a = t2.a
    UNION
    SELECT * FROM test1 t1, test2 t2 WHERE t1.a = t2.b
);
```

The optimized SQL query consists of two equivalent join subqueries, and each subquery can be used for hash join in this scenario. The optimized execution plan is as follows:

```
                                                            QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------
---
 HashAggregate  (cost=1634.99..2096.81 rows=46182 width=16) (actual time=6.369..6.772 rows=1000
loops=1)
   Group By Key: t1.a, t1.b, t2.a, t2.b
   ->  Append  (cost=58.35..1173.17 rows=46182 width=16) (actual time=0.833..3.414 rows=2000 loops=1)
         ->  Hash Join  (cost=58.35..355.67 rows=23091 width=16) (actual time=0.832..1.590 rows=1000
loops=1)
               Hash Cond: (t1.a = t2.a)
               ->  Seq Scan on test1 t1  (cost=0.00..31.49 rows=2149 width=8) (actual time=0.015..0.156
rows=1000 loops=1)
               ->  Hash  (cost=31.49..31.49 rows=2149 width=8) (actual time=0.531..0.531 rows=1000 loops=1)
                     Buckets: 32768  Batches: 1  Memory Usage: 40kB
                     ->  Seq Scan on test2 t2  (cost=0.00..31.49 rows=2149 width=8) (actual time=0.010..0.199
rows=1000 loops=1)
         ->  Hash Join  (cost=58.35..355.67 rows=23091 width=16) (actual time=0.694..1.421 rows=1000
loops=1)
               Hash Cond: (t1.a = t2.b)
               ->  Seq Scan on test1 t1  (cost=0.00..31.49 rows=2149 width=8) (actual time=0.010..0.160
rows=1000 loops=1)
               ->  Hash  (cost=31.49..31.49 rows=2149 width=8) (actual time=0.524..0.524 rows=1000 loops=1)
                     Buckets: 32768  Batches: 1  Memory Usage: 40kB
                     ->  Seq Scan on test2 t2  (cost=0.00..31.49 rows=2149 width=8) (actual time=0.008..0.177
rows=1000 loops=1)
 Total runtime: 7.759 ms
(16 rows)
```

10 Configuring Running Parameters

10.1 Viewing Parameter Values

GaussDB uses a set of default running parameters after it is installed. You can modify the parameters to better fit the current service scenarios and data volume.

Procedure

Step 1 Connect to a database. For details, see **Connecting to a Database**.

Step 2 View the parameter values in the database.

- Method 1: Run the **SHOW** command.
  - Run the following command to view the value of a certain parameter:
    ```
    openGauss=# SHOW server_version;
    ```
    server_version indicates the database version.
  - Run the following command to view values of all parameters:
    ```
    openGauss=#  SHOW ALL;
    ```
- Method 2: Query the **pg_settings** view.
  - Run the following command to view the value of a certain parameter:
    ```
    openGauss=# SELECT * FROM pg_settings WHERE NAME='server_version';
    ```
  - Run the following command to view values of all parameters:
    ```
    openGauss=# SELECT * FROM pg_settings;
    ```

----End

Example

View the server version.

```
openGauss=# SHOW server_version;
 server_version
----------------
 9.2.4
(1 row)
```

10.2 Resetting Parameters

You are advised to modify some parameters on the GaussDB console. If the parameters cannot be modified on the console, evaluate the risks and contact customer service.

Background

GaussDB provides multiple methods to set the GUC parameters of databases, users, or sessions.

- Parameter names are case-insensitive.
- A parameter value can be an integer, floating point value, string, Boolean value, or enumerated value.
  - The Boolean values can be **on**/**off**, **true**/**false**, **yes**/**no**, or **1**/**0**, and are case-insensitive.
  - The enumerated value range is specified in the **enumvals** column of the **pg_settings** system catalog.
- For parameters using units, specify their units during the setting. Otherwise, default units are used.
  - The default units are specified in the **unit** column of **pg_settings**.
  - The unit of memory can be KB, MB, or GB.
  - The unit of time can be ms, s, min, h, or d.

For details about parameters in the hosts configuration template, see GaussDB.

Setting GUC Parameters

GaussDB provides six types of GUC parameters. For details about parameter types and their setting methods, see **Table 10-1**.

Table 10-1 GUC parameters

Category	Description	How to Set
INTERNAL	Fixed parameter. It is set during database creation and cannot be modified. Users can only view the parameter by running the **SHOW** command or in the **pg_settings** view.	None
POSTMASTER	Database server parameter. It can be set when the database is started or in the configuration file.	Method 1 in **Table 10-2**.
SIGHUP	Global database parameter. It can be set when the database is started or be modified later.	Method 1, 2, in **Table 10-2**.

Category	Description	How to Set
BACKEND	Session connection parameter. It is specified during session connection creation and cannot be modified after that. The parameter setting becomes invalid when the session is disconnected. This is an internal parameter and not recommended for users to set it.	Method 1, 2, in **Table 10-2**. **NOTE** The parameter setting takes effect when the next session is created.
SUSET	Database administrator parameter. It can be set by common users when or after the database is started. It can also be set by database administrators using SQL statements.	Method 1 or 2 by a common user, or method 3 by a database administrator in **Table 10-2**.
USERSET	Common user parameter. It can be set by any user at any time.	Method 1, 2, or 3 in **Table 10-2**.

You can set GUC parameters in GaussDB using the three methods listed in **Table 10-2**.

Table 10-2 Methods for setting GUC parameters

No.	How to Set
Method 1	1. Log in to the management console. 2. On the **Instances** page, click the instance name to go to the **Basic Information** page. 3. In the navigation pane on the left, click **Parameters**. On the displayed page, modify parameters. If the parameters cannot be modified on the console, evaluate the risks in advance and contact customer service for modification. 4. Restart the database to make the setting take effect. **NOTE** Rebooting instances will interrupt user operations. Plan a proper execution window before the restart.
Method 2	1. Log in to the management console. 2. On the **Instances** page, click the instance name to go to the **Basic Information** page. 3. In the navigation pane on the left, click **Parameters**. On the displayed page, modify parameters. If the parameters cannot be modified on the console, evaluate the risks in advance and contact customer service for modification.

No.	How to Set

Method 3:	Set parameters at specific database, user, and session levels.
	● Set a database-level parameter.
	`openGauss=# ALTER DATABASE dbname SET paraname TO value;`
	The setting takes effect in the next session.
	● Set a user-level parameter.
	`openGauss=# ALTER USER username SET paraname TO value;`
	The setting takes effect in the next session.
	● Set a session-level parameter.
	`openGauss=# SET paraname TO value;`
	Parameter value in the current session is changed. After you exit the session, the setting becomes invalid.
	NOTE
	Session-level parameters set by **SET** have the highest priority, followed by parameters set by **ALTER**. Parameter values set by **ALTER DATABASE** have a higher priority than those set using **ALTER USER**. Priorities of the first three methods are all higher than those of **gs_guc**.

CAUTION

If you use method 1 or 2 to set a parameter that does not belong to the current environment, the database displays a message indicating that the parameter is not supported.

11 SQL Reference

11.1 SQL

What Is SQL?

SQL is a standard computer language used to control the access to databases and manage data in databases.

SQL provides different statements to enable you to:

- Query data.
- Insert, update, and delete rows.
- Create, replace, modify, and delete objects.
- Control the access to a database and its objects.
- Maintain the consistency and integrity of a database.

SQL consists of commands and functions that are used to manage databases and database objects. SQL can also forcibly implement the rules for data types, expressions, and texts. Therefore, **SQL Reference** describes data types, expressions, functions, and operators in addition to SQL syntax.

Development of SQL Standards

The development history of SQL standards is as follows:

- 1986: ANSI X3.135-1986, ISO/IEC 9075:1986, SQL-86
- 1989: ANSI X3.135-1989, ISO/IEC 9075:1989, SQL-89
- 1992: ANSI X3.135-1992, ISO/IEC 9075:1992, SQL-92 (SQL2)
- 1999: ISO/IEC 9075:1999, SQL:1999 (SQL3)
- 2003: ISO/IEC 9075:2003, SQL:2003 (SQL4)
- 2011: ISO/IEC 9075:200N, SQL:2011 (SQL5)

SQL Standards Supported by GaussDB

GaussDB supports major SQL2, SQL3, and SQL4 features by default.

11.2 Keywords

The SQL contains reserved words and non-reserved words. Standards require that reserved keywords not be used as other identifiers. Non-reserved keywords have special meanings only in a specific environment and can be used as identifiers in other environments.

The naming rules for identifiers are as follows:

- An identifier name can only contain letters, underscores, digits (0-9), and dollar signs ($).
- An identifier name must start with a letter (a to z) or an underscore (_).

NOTE

- The naming rules are recommended but not mandatory.
- In special cases, double quotation marks (") can be used to avoid special character errors.

Table 11-1 SQL keywords

Keyword	GaussDB	SQL:1999	SQL-92
ABORT	Non-reserved	N/A	N/A
ABS	N/A	Non-reserved	N/A
ABSOLUTE	Non-reserved	Reserved	Reserved
ACCESS	Non-reserved	N/A	N/A
ACCOUNT	Non-reserved	N/A	N/A
ACTION	Non-reserved	Reserved	Reserved
ADA	N/A	Non-reserved	Non-reserved
ADD	Non-reserved	Reserved	Reserved
ADMIN	Non-reserved	Reserved	N/A
AFTER	Non-reserved	Reserved	N/A
AGGREGATE	Non-reserved	Reserved	N/A
ALGORITHM	Non-reserved	N/A	N/A
ALIAS	N/A	Reserved	N/A
ALL	Reserved	Reserved	Reserved
ALLOCATE	N/A	Reserved	Reserved
ALSO	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
ALTER	Non-reserved	Reserved	Reserved
ALWAYS	Non-reserved	N/A	N/A
ANALYSE	Reserved	N/A	N/A
ANALYZE	Reserved	N/A	N/A
AND	Reserved	Reserved	Reserved
ANY	Reserved	Reserved	Reserved
APP	Non-reserved	N/A	N/A
APPEND	Non-reserved	N/A	N/A
ARCHIVE	Non-reserved	N/A	N/A
ARE	N/A	Reserved	Reserved
ARRAY	Reserved	Reserved	N/A
AS	Reserved	Reserved	Reserved
ASC	Reserved	Reserved	Reserved
ASENSITIVE	N/A	Non-reserved	N/A
ASSERTION	Non-reserved	Reserved	Reserved
ASSIGNMENT	Non-reserved	Non-reserved	N/A
ASYMMETRIC	Reserved	Non-reserved	N/A
AT	Non-reserved	Reserved	Reserved
ATOMIC	N/A	Non-reserved	N/A
ATTRIBUTE	Non-reserved	N/A	N/A
AUDIT	Non-reserved	N/A	N/A
AUTHID	Reserved	N/A	N/A
AUTHORIZATION	Reserved (functions and types allowed)	Reserved	Reserved
AUTOEXTEND	Non-reserved	N/A	N/A
AUTOMAPPED	Non-reserved	N/A	N/A
AVG	N/A	Non-reserved	Reserved

Keyword	GaussDB	SQL:1999	SQL-92
BACKWARD	Non-reserved	N/A	N/A
BARRIER	Non-reserved	N/A	N/A
BEFORE	Non-reserved	Reserved	N/A
BEGIN	Non-reserved	Reserved	Reserved
BEGIN_NON_ANOYBLOCK	Non-reserved	N/A	N/A
BETWEEN	Non-reserved (excluding functions and types)	Non-reserved	Reserved
BIGINT	Non-reserved (excluding functions and types)	N/A	N/A
BINARY	Reserved (functions and types allowed)	Reserved	N/A
BINARY_DOUBLE	Non-reserved (excluding functions and types)	N/A	N/A
BINARY_INTEGER	Non-reserved (excluding functions and types)	N/A	N/A
BIT	Non-reserved (excluding functions and types)	Reserved	Reserved
BITVAR	N/A	Non-reserved	N/A
BIT_LENGTH	N/A	Non-reserved	Reserved
BLANKS	Non-reserved	N/A	N/A
BLOB	Non-reserved	Reserved	N/A
BLOCKCHAIN	Non-reserved	N/A	N/A
BODY	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
BOOLEAN	Non-reserved (excluding functions and types)	Reserved	N/A
BOTH	Reserved	Reserved	Reserved
BUCKETCNT	Non-reserved (excluding functions and types)	N/A	N/A
BUCKETS	Reserved	N/A	N/A
BREADTH	N/A	Reserved	N/A
BY	Non-reserved	Reserved	Reserved
BYTEAWITHOUTORDER	Non-reserved (excluding functions and types)	N/A	N/A
BYTEAWITHOUTORDER-WITHEQUAL	Non-reserved (excluding functions and types)	N/A	N/A
C	N/A	Non-reserved	Non-reserved
CACHE	Non-reserved	N/A	N/A
CALL	Non-reserved	Reserved	N/A
CALLED	Non-reserved	Non-reserved	N/A
CANCELABLE	Non-reserved	N/A	N/A
CARDINALITY	N/A	Non-reserved	N/A
CASCADE	Non-reserved	Reserved	Reserved
CASCADED	Non-reserved	Reserved	Reserved
CASE	Reserved	Reserved	Reserved
CAST	Reserved	Reserved	Reserved
CATALOG	Non-reserved	Reserved	Reserved
CATALOG_NAME	N/A	Non-reserved	Non-reserved

Keyword	GaussDB	SQL:1999	SQL-92
CHAIN	Non-reserved	Non-reserved	N/A
CHAR	Non-reserved (excluding functions and types)	Reserved	Reserved
CHARACTER	Non-reserved (excluding functions and types)	Reserved	Reserved
CHARACTERISTICS	Non-reserved	N/A	N/A
CHARACTERSET	Non-reserved	N/A	N/A
CHARACTER_LENGTH	N/A	Non-reserved	Reserved
CHARACTER_SET_CATALOG	N/A	Non-reserved	Non-reserved
CHARACTER_SET_NAME	N/A	Non-reserved	Non-reserved
CHARACTER_SET_SCHEMA	N/A	Non-reserved	Non-reserved
CHAR_LENGTH	N/A	Non-reserved	Reserved
CHECK	Reserved	Reserved	Reserved
CHECKED	N/A	Non-reserved	N/A
CHECKPOINT	Non-reserved	N/A	N/A
CLASS	Non-reserved	Reserved	N/A
CLEAN	Non-reserved	N/A	N/A
CLASS_ORIGIN	N/A	Non-reserved	Non-reserved
CLIENT	Non-reserved	N/A	N/A
CLIENT_MASTER_KEY	Non-reserved	N/A	N/A
CLIENT_MASTER_KEYS	Non-reserved	N/A	N/A
CLOB	Non-reserved	Reserved	N/A
CLOSE	Non-reserved	Reserved	Reserved
CLUSTER	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
COALESCE	Non-reserved (excluding functions and types)	Non-reserved	Reserved
COBOL	N/A	Non-reserved	Non-reserved
COLLATE	Reserved	Reserved	Reserved
COLLATION	Reserved (functions and types allowed)	Reserved	Reserved
COLLATION_CATALOG	N/A	Non-reserved	Non-reserved
COLLATION_NAME	N/A	Non-reserved	Non-reserved
COLLATION_SCHEMA	N/A	Non-reserved	Non-reserved
COLUMN	Reserved	Reserved	Reserved
COLUMN_ENCRYPTION_KEY	Non-reserved	N/A	N/A
COLUMN_ENCRYPTION_KEYS	Non-reserved	N/A	N/A
COLUMN_NAME	N/A	Non-reserved	Non-reserved
COMPACT	Reserved (functions and types allowed)	N/A	N/A
COMPATIBLE_ILLEGAL_CHARS	Non-reserved	N/A	N/A
COMMAND_FUNCTION	N/A	Non-reserved	Non-reserved
COMPLETE	Non-reserved	N/A	N/A
COMMAND_FUNCTION_CODE	N/A	Non-reserved	N/A
COMMENT	Non-reserved	N/A	N/A
COMMENTS	Non-reserved	N/A	N/A
COMMIT	Non-reserved	Reserved	Reserved

Keyword	GaussDB	SQL:1999	SQL-92
COMMITTED	Non-reserved	Non-reserved	Non-reserved
COMPRESS	Non-reserved	N/A	N/A
COMPLETION	N/A	Reserved	N/A
CONCURRENTLY	Reserved (functions and types allowed)	N/A	N/A
CONDITION	Non-reserved	N/A	N/A
CONDITION_NUMBER	N/A	Non-reserved	Non-reserved
CONFIGURATION	Non-reserved	N/A	N/A
CONNECT	Non-reserved	Reserved	Reserved
CONNECTION	Non-reserved	Reserved	Reserved
CONNECTION_NAME	N/A	Non-reserved	Non-reserved
CONSTANT	Non-reserved	N/A	N/A
CONSTRAINT	Reserved	Reserved	Reserved
CONSTRAINTS	Non-reserved	Reserved	Reserved
CONSTRAINT_CATALOG	N/A	Non-reserved	Non-reserved
CONSTRAINT_NAME	N/A	Non-reserved	Non-reserved
CONSTRAINT_SCHEMA	N/A	Non-reserved	Non-reserved
CONSTRUCTOR	N/A	Reserved	N/A
CONTAINS	N/A	Non-reserved	N/A
CONTENT	Non-reserved	N/A	N/A
CONTINUE	Non-reserved	Reserved	Reserved
CONTVIEW	Non-reserved	N/A	N/A
CONVERSION	Non-reserved	N/A	N/A
CONVERT	N/A	Non-reserved	Reserved
COORDINATOR	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
COORDINATORS	Non-reserved	N/A	N/A
COPY	Non-reserved	N/A	N/A
CORRESPONDING	N/A	Reserved	Reserved
COST	Non-reserved	N/A	N/A
COUNT	N/A	Non-reserved	Reserved
CREATE	Reserved	Reserved	Reserved
CROSS	Reserved (functions and types allowed)	Reserved	Reserved
CSN	Reserved (functions and types allowed)	N/A	N/A
CSV	Non-reserved	N/A	N/A
CUBE	Non-reserved	Reserved	N/A
CURRENT	Non-reserved	Reserved	Reserved
CURRENT_CATALOG	Reserved	N/A	N/A
CURRENT_DATE	Reserved	Reserved	Reserved
CURRENT_PATH	N/A	Reserved	N/A
CURRENT_ROLE	Reserved	Reserved	N/A
CURRENT_SCHEMA	Reserved (functions and types allowed)	N/A	N/A
CURRENT_TIME	Reserved	Reserved	Reserved
CURRENT_TIMESTAMP	Reserved	Reserved	Reserved
CURRENT_USER	Reserved	Reserved	Reserved
CURSOR	Non-reserved	Reserved	Reserved
CURSOR_NAME	N/A	Non-reserved	Non-reserved
CYCLE	Non-reserved	Reserved	N/A
DATA	Non-reserved	Reserved	Non-reserved
DATABASE	Non-reserved	N/A	N/A
DATAFILE	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
DATANODE	Non-reserved	N/A	N/A
DATANODES	Non-reserved	N/A	N/A
DATE_FORMAT	Non-reserved	N/A	N/A
DATATYPE_CL	Non-reserved	N/A	N/A
DATE	Non-reserved (excluding functions and types)	Reserved	Reserved
DELTAMERGE	Reserved (functions and types allowed)	N/A	N/A
DATETIME_INTERVAL_CODE	N/A	Non-reserved	Non-reserved
DATETIME_INTERVAL_PRECISION	N/A	Non-reserved	Non-reserved
DAY	Non-reserved	Reserved	Reserved
DBCOMPATIBILITY	Non-reserved	N/A	N/A
DEALLOCATE	Non-reserved	Reserved	Reserved
DEC	Non-reserved (excluding functions and types)	Reserved	Reserved
DECIMAL	Non-reserved (excluding functions and types)	Reserved	Reserved
DECLARE	Non-reserved	Reserved	Reserved
DECODE	Non-reserved (excluding functions and types)	N/A	N/A
DEFAULT	Reserved	Reserved	Reserved
DEFAULTS	Non-reserved	N/A	N/A
DEFERRABLE	Reserved	Reserved	Reserved
DEFERRED	Non-reserved	Reserved	Reserved
DEFINED	N/A	Non-reserved	N/A

Keyword	GaussDB	SQL:1999	SQL-92
DEFINER	Non-reserved	Non-reserved	N/A
DELETE	Non-reserved	Reserved	Reserved
DELIMITER	Non-reserved	N/A	N/A
DELIMITERS	Non-reserved	N/A	N/A
DELTA	Non-reserved	N/A	N/A
DEPTH	N/A	Reserved	N/A
DEREF	N/A	Reserved	N/A
DESC	Reserved	Reserved	Reserved
DESCRIBE	N/A	Reserved	Reserved
DESCRIPTOR	N/A	Reserved	Reserved
DESTROY	N/A	Reserved	N/A
DESTRUCTOR	N/A	Reserved	N/A
DETERMINISTIC	Non-reserved	Reserved	N/A
DIAGNOSTICS	N/A	Reserved	Reserved
DICTIONARY	Non-reserved	Reserved	N/A
DIRECT	Non-reserved	N/A	N/A
DIRECTORY	Non-reserved	N/A	N/A
DISABLE	Non-reserved	N/A	N/A
DISCARD	Non-reserved	N/A	N/A
DISCONNECT	Non-reserved	Reserved	Reserved
DISPATCH	N/A	Non-reserved	N/A
DISTINCT	Reserved	Reserved	Reserved
DISTRIBUTE	Non-reserved	N/A	N/A
DISTRIBUTION	Non-reserved	N/A	N/A
DO	Reserved	N/A	N/A
DOCUMENT	Non-reserved	N/A	N/A
DOMAIN	Non-reserved	Reserved	Reserved
DOUBLE	Non-reserved	Reserved	Reserved
DROP	Non-reserved	Reserved	Reserved

Keyword	GaussDB	SQL:1999	SQL-92
DUPLICATE	Non-reserved	N/A	N/A
DYNAMIC	N/A	Reserved	N/A
DYNAMIC_FUNCTION	N/A	Non-reserved	Non-reserved
DYNAMIC_FUNCTION_CODE	N/A	Non-reserved	N/A
EACH	Non-reserved	Reserved	N/A
ELSE	Reserved	Reserved	Reserved
ELASTIC	Non-reserved	N/A	N/A
ENABLE	Non-reserved	N/A	N/A
ENCLOSED	Non-reserved	N/A	N/A
ENCODING	Non-reserved	N/A	N/A
ENCRYPTED	Non-reserved	N/A	N/A
ENCRYPTED_VALUE	Non-reserved	N/A	N/A
ENCRYPTION	Non-reserved	N/A	N/A
ENCRYPTION_TYPE	Non-reserved	N/A	N/A
END	Reserved	Reserved	Reserved
END-EXEC	N/A	Reserved	Reserved
ENFORCED	Non-reserved	N/A	N/A
ENUM	Non-reserved	N/A	N/A
EOL	Non-reserved	N/A	N/A
ERRORS	Non-reserved	N/A	N/A
EQUALS	N/A	Reserved	N/A
ESCAPE	Non-reserved	Reserved	Reserved
ESCAPING	Non-reserved	N/A	N/A
EVERY	Non-reserved	Reserved	N/A
EXCEPT	Reserved	Reserved	Reserved
EXCEPTION	N/A	Reserved	Reserved
EXCHANGE	Non-reserved	N/A	N/A
EXCLUDE	Non-reserved	N/A	N/A
EXCLUDED	Reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
EXCLUDING	Non-reserved	N/A	N/A
EXCLUSIVE	Non-reserved	N/A	N/A
EXEC	N/A	Reserved	Reserved
EXECUTE	Non-reserved	Reserved	Reserved
EXISTING	N/A	Non-reserved	N/A
EXISTS	Non-reserved (excluding functions and types)	Non-reserved	Reserved
EXPIRED_P	Non-reserved	N/A	N/A
EXPLAIN	Non-reserved	N/A	N/A
EXTENSION	Non-reserved	N/A	N/A
EXTERNAL	Non-reserved	Reserved	Reserved
EXTRACT	Non-reserved (excluding functions and types)	Non-reserved	Reserved
FALSE	Reserved	Reserved	Reserved
FAMILY	Non-reserved	N/A	N/A
FAST	Non-reserved	N/A	N/A
FEATURES	Non-reserved	N/A	N/A
FETCH	Reserved	Reserved	Reserved
FENCED	Reserved	N/A	N/A
FIELDS	Non-reserved	N/A	N/A
FILEHEADER	Non-reserved	N/A	N/A
FILLER	Non-reserved	N/A	N/A
FILTER	Non-reserved	Reserved	Reserved
FINAL	N/A	Non-reserved	N/A
FIRST	Non-reserved	Reserved	Reserved
FIXED	Non-reserved	Reserved	Reserved
FILL_MISSING_FIELDS	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
FLOAT	Non-reserved (excluding functions and types)	Reserved	Reserved
FOLLOWING	Non-reserved	N/A	N/A
FOR	Reserved	Reserved	Reserved
FORCE	Non-reserved	N/A	N/A
FOREIGN	Reserved	Reserved	Reserved
FORMATTER	Non-reserved	N/A	N/A
FORTRAN	N/A	Non-reserved	Non-reserved
FORWARD	Non-reserved	N/A	N/A
FOUND	N/A	Reserved	Reserved
FREE	N/A	Reserved	N/A
FREEZE	Reserved (functions and types allowed)	N/A	N/A
FROM	Reserved	Reserved	Reserved
FULL	Reserved (functions and types allowed)	Reserved	Reserved
FUNCTION	Non-reserved	Reserved	N/A
FUNCTIONS	Non-reserved	N/A	N/A
G	N/A	Non-reserved	N/A
GENERAL	N/A	Reserved	N/A
GENERATED	Non-reserved	Non-reserved	N/A
GET	N/A	Reserved	Reserved
GLOBAL	Non-reserved	Reserved	Reserved
GO	N/A	Reserved	Reserved
GOTO	N/A	Reserved	Reserved
GRANT	Reserved	Reserved	Reserved

Keyword	GaussDB	SQL:1999	SQL-92
GRANTED	Non-reserved	Non-reserved	N/A
GREATEST	Non-reserved (excluding functions and types)	N/A	N/A
GROUP	Reserved	Reserved	Reserved
GROUPING	Non-reserved (excluding functions and types)	Reserved	N/A
GROUPPARENT	Reserved	N/A	N/A
HANDLER	Non-reserved	N/A	N/A
HAVING	Reserved	Reserved	Reserved
HDFSDIRECTORY	Reserved (functions and types allowed)	N/A	N/A
HEADER	Non-reserved	N/A	N/A
HIERARCHY	N/A	Non-reserved	N/A
HOLD	Non-reserved	Non-reserved	N/A
HOST	N/A	Reserved	N/A
HOUR	Non-reserved	Reserved	Reserved
IDENTIFIED	Non-reserved	N/A	N/A
IDENTITY	Non-reserved	Reserved	Reserved
IF	Non-reserved	N/A	N/A
IGNORE	N/A	Reserved	N/A
IGNORE_EXTRA_DATA	Non-reserved	N/A	N/A
ILIKE	Reserved (functions and types allowed)	N/A	N/A
IMMEDIATE	Non-reserved	Reserved	Reserved
IMMUTABLE	Non-reserved	N/A	N/A
IMPLEMENTATION	N/A	Non-reserved	N/A

Keyword	GaussDB	SQL:1999	SQL-92
IMPLICIT	Non-reserved	N/A	N/A
IN	Reserved	Reserved	Reserved
INTERNAL	Non-reserved	N/A	N/A
INCLUDE	Non-reserved	N/A	N/A
INCLUDING	Non-reserved	N/A	N/A
INCREMENT	Non-reserved	N/A	N/A
INCREMENTAL	Non-reserved	N/A	N/A
INDEX	Non-reserved	N/A	N/A
INDEXES	Non-reserved	N/A	N/A
INDICATOR	N/A	Reserved	Reserved
INFILE	Non-reserved	N/A	N/A
INFIX	N/A	Non-reserved	N/A
INHERIT	Non-reserved	N/A	N/A
INHERITS	Non-reserved	N/A	N/A
INITIAL	Non-reserved	N/A	N/A
INITIALIZE	N/A	Reserved	N/A
INITIALLY	Reserved	Reserved	Reserved
INITRANS	Non-reserved	N/A	N/A
INLINE	Non-reserved	N/A	N/A
INNER	Reserved (functions and types allowed)	Reserved	Reserved
INOUT	Non-reserved (excluding functions and types)	Reserved	N/A
INPUT	Non-reserved	Reserved	Reserved
INSENSITIVE	Non-reserved	Non-reserved	Reserved
INSERT	Non-reserved	Reserved	Reserved
INSTANCE	N/A	Non-reserved	N/A

Keyword	GaussDB	SQL:1999	SQL-92
INSTANTIABLE	N/A	Non-reserved	N/A
INSTEAD	Non-reserved	N/A	N/A
INT	Non-reserved (excluding functions and types)	Reserved	Reserved
INTEGER	Non-reserved (excluding functions and types)	Reserved	Reserved
INTERSECT	Reserved	Reserved	Reserved
INTERVAL	Non-reserved (excluding functions and types)	Reserved	Reserved
INTO	Reserved	Reserved	Reserved
INVOKER	Non-reserved	Non-reserved	N/A
IP	Non-reserved	N/A	N/A
IS	Reserved	Reserved	Reserved
ISNULL	Non-reserved	N/A	N/A
ISOLATION	Non-reserved	Reserved	Reserved
ITERATE	N/A	Reserved	N/A
JOIN	Reserved (functions and types allowed)	Reserved	Reserved
K	N/A	Non-reserved	N/A
KEY	Non-reserved	Reserved	Reserved
KEY_PATH	Non-reserved	N/A	N/A
KEY_MEMBER	N/A	Non-reserved	N/A
KEY_STORE	Non-reserved	N/A	N/A
KEY_TYPE	N/A	Non-reserved	N/A
KILL	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
LABEL	Non-reserved	N/A	N/A
LANGUAGE	Non-reserved	Reserved	Reserved
LARGE	Non-reserved	Reserved	N/A
LAST	Non-reserved	Reserved	Reserved
LATERAL	N/A	Reserved	N/A
LC_COLLATE	Non-reserved	N/A	N/A
LC_CTYPE	Non-reserved	N/A	N/A
LEADING	Reserved	Reserved	Reserved
LEAKPROOF	Non-reserved	N/A	N/A
LEAST	Non-reserved (excluding functions and types)	N/A	N/A
LEFT	Reserved (functions and types allowed)	Reserved	Reserved
LENGTH	N/A	Non-reserved	Non-reserved
LESS	Reserved	Reserved	N/A
LEVEL	Non-reserved	Reserved	Reserved
LIKE	Reserved (functions and types allowed)	Reserved	Reserved
LIMIT	Reserved	Reserved	N/A
LIST	Non-reserved	N/A	N/A
LISTEN	Non-reserved	N/A	N/A
LOAD	Non-reserved	N/A	N/A
LOCAL	Non-reserved	Reserved	Reserved
LOCALTIME	Reserved	Reserved	N/A
LOCALTIMESTAMP	Reserved	Reserved	N/A
LOCATION	Non-reserved	N/A	N/A
LOCATOR	N/A	Reserved	N/A
LOCK	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
LOG	Non-reserved	N/A	N/A
LOGGING	Non-reserved	N/A	N/A
LOGIN_ANY	Non-reserved	N/A	N/A
LOGIN_FAILURE	Non-reserved	N/A	N/A
LOGIN_SUCCESS	Non-reserved	N/A	N/A
LOGOUT	Non-reserved	N/A	N/A
LOOP	Non-reserved	N/A	N/A
LOWER	N/A	Non-reserved	Reserved
MAP	N/A	Reserved	N/A
MAPPING	Non-reserved	N/A	N/A
MASKING	Non-reserved	N/A	N/A
MASTER	Non-reserved	N/A	N/A
MATCH	Non-reserved	Reserved	Reserved
MATCHED	Non-reserved	N/A	N/A
MATERIALIZED	Non-reserved	N/A	N/A
MAX	N/A	Non-reserved	Reserved
MAXEXTENTS	Non-reserved	N/A	N/A
MAXSIZE	Non-reserved	N/A	N/A
MAXTRANS	Non-reserved	N/A	N/A
MAXVALUE	Reserved	N/A	N/A
MERGE	Non-reserved	N/A	N/A
MESSAGE_LENGTH	N/A	Non-reserved	Non-reserved
MESSAGE_OCTET_LENGTH	N/A	Non-reserved	Non-reserved
MESSAGE_TEXT	N/A	Non-reserved	Non-reserved
METHOD	N/A	Non-reserved	N/A
MIN	N/A	Non-reserved	Reserved

Keyword	GaussDB	SQL:1999	SQL-92
MINEXTENTS	Non-reserved	N/A	N/A
MINUS	Reserved	N/A	N/A
MINUTE	Non-reserved	Reserved	Reserved
MINVALUE	Non-reserved	N/A	N/A
MOD	N/A	Non-reserved	N/A
MODE	Non-reserved	N/A	N/A
MODEL	Non-reserved	N/A	N/A
MODIFIES	N/A	Reserved	N/A
MODIFY	Reserved	Reserved	N/A
MODULE	N/A	Reserved	Reserved
MONTH	Non-reserved	Reserved	Reserved
MORE	N/A	Non-reserved	Non-reserved
MOVE	Non-reserved	N/A	N/A
MOVEMENT	Non-reserved	N/A	N/A
MUMPS	N/A	Non-reserved	Non-reserved
NAME	Non-reserved	Non-reserved	Non-reserved
NAMES	Non-reserved	Reserved	Reserved
NATIONAL	Non-reserved (excluding functions and types)	Reserved	Reserved
NATURAL	Reserved (functions and types allowed)	Reserved	Reserved
NCHAR	Non-reserved (excluding functions and types)	Reserved	Reserved
NCLOB	N/A	Reserved	N/A
NEW	N/A	Reserved	N/A
NEXT	Non-reserved	Reserved	Reserved

Keyword	GaussDB	SQL:1999	SQL-92
NO	Non-reserved	Reserved	Reserved
NOCOMPRESS	Non-reserved	N/A	N/A
NOCYCLE	Non-reserved	N/A	N/A
NODE	Non-reserved	N/A	N/A
NOLOGGING	Non-reserved	N/A	N/A
NOMAXVALUE	Non-reserved	N/A	N/A
NOMINVALUE	Non-reserved	N/A	N/A
NONE	Non-reserved (excluding functions and types)	Reserved	N/A
NOT	Reserved	Reserved	Reserved
NOTHING	Non-reserved	N/A	N/A
NOTIFY	Non-reserved	N/A	N/A
NOTNULL	Reserved (functions and types allowed)	N/A	N/A
NOWAIT	Non-reserved	N/A	N/A
NULL	Reserved	Reserved	Reserved
NULLABLE	N/A	Non-reserved	Non-reserved
NULLCOLS	Non-reserved	N/A	N/A
NULLIF	Non-reserved (excluding functions and types)	Non-reserved	Reserved
NULLS	Non-reserved	N/A	N/A
NUMBER	Non-reserved (excluding functions and types)	Non-reserved	Non-reserved
NUMERIC	Non-reserved (excluding functions and types)	Reserved	Reserved
NUMSTR	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
NVARCHAR	Non-reserved (excluding functions and types)	N/A	N/A
NVARCHAR2	Non-reserved (excluding functions and types)	N/A	N/A
NVL	Non-reserved (excluding functions and types)	N/A	N/A
OBJECT	Non-reserved	Reserved	N/A
OCTET_LENGTH	N/A	Non-reserved	Reserved
OF	Non-reserved	Reserved	Reserved
OFF	Non-reserved	Reserved	N/A
OFFSET	Reserved	N/A	N/A
OIDS	Non-reserved	N/A	N/A
OLD	N/A	Reserved	N/A
ON	Reserved	Reserved	Reserved
ONLY	Reserved	Reserved	Reserved
OPEN	N/A	Reserved	Reserved
OPERATION	N/A	Reserved	N/A
OPERATOR	Non-reserved	N/A	N/A
OPTIMIZATION	Non-reserved	N/A	N/A
OPTION	Non-reserved	Reserved	Reserved
OPTIONALLY	Non-reserved	N/A	N/A
OPTIONS	Non-reserved	Non-reserved	N/A
OR	Reserved	Reserved	Reserved
ORDER	Reserved	Reserved	Reserved
ORDINALITY	N/A	Reserved	N/A

Keyword	GaussDB	SQL:1999	SQL-92
OUT	Non-reserved (excluding functions and types)	Reserved	N/A
OUTER	Reserved (functions and types allowed)	Reserved	Reserved
OUTPUT	N/A	Reserved	Reserved
OVER	Non-reserved	N/A	N/A
OVERLAPS	Reserved (functions and types allowed)	Non-reserved	Reserved
OVERLAY	Non-reserved (excluding functions and types)	Non-reserved	N/A
OVERRIDING	N/A	Non-reserved	N/A
OWNED	Non-reserved	N/A	N/A
OWNER	Non-reserved	N/A	N/A
PACKAGE	Non-reserved	N/A	N/A
PACKAGES	Non-reserved	N/A	N/A
PAD	N/A	Reserved	Reserved
PARAMETER	N/A	Reserved	N/A
PARAMETERS	N/A	Reserved	N/A
PARAMETER_MODE	N/A	Non-reserved	N/A
PARAMETER_NAME	N/A	Non-reserved	N/A
PARAMETER_ORDINAL_POSITION	N/A	Non-reserved	N/A
PARAMETER_SPECIFIC_CATALOG	N/A	Non-reserved	N/A
PARAMETER_SPECIFIC_NAME	N/A	Non-reserved	N/A
PARAMETER_SPECIFIC_SCHEMA	N/A	Non-reserved	N/A

Keyword	GaussDB	SQL:1999	SQL-92
PARSER	Non-reserved	N/A	N/A
PARTIAL	Non-reserved	Reserved	Reserved
PARTITION	Non-reserved	N/A	N/A
PARTITIONS	Non-reserved	N/A	N/A
PASCAL	N/A	Non-reserved	Non-reserved
PASSING	Non-reserved	N/A	N/A
PASSWORD	Non-reserved	N/A	N/A
PATH	N/A	Reserved	N/A
PCTFREE	Non-reserved	N/A	N/A
PER	Non-reserved	N/A	N/A
PERM	Non-reserved	N/A	N/A
PERCENT	Non-reserved	N/A	N/A
PERFORMANCE	Reserved	N/A	N/A
PLACING	Reserved	N/A	N/A
PLAN	Non-reserved	N/A	N/A
PLANS	Non-reserved	N/A	N/A
PLI	N/A	Non-reserved	Non-reserved
POOL	Non-reserved	N/A	N/A
POLICY	Non-reserved	N/A	N/A
POSITION	Non-reserved (excluding functions and types)	Non-reserved	Reserved
POSTFIX	N/A	Reserved	N/A
PRECEDING	Non-reserved	N/A	N/A
PRECISION	Non-reserved (excluding functions and types)	Reserved	Reserved
PREDICT	Non-reserved	N/A	N/A
PREFERRED	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
PREFIX	Non-reserved	Reserved	N/A
PREORDER	N/A	Reserved	N/A
PREPARE	Non-reserved	Reserved	Reserved
PREPARED	Non-reserved	N/A	N/A
PRESERVE	Non-reserved	Reserved	Reserved
PRIMARY	Reserved	Reserved	Reserved
PRIOR	Non-reserved	Reserved	Reserved
PRIORER	Reserved	N/A	N/A
PRIVATE	Non-reserved	N/A	N/A
PRIVILEGE	Non-reserved	N/A	N/A
PRIVILEGES	Non-reserved	Reserved	Reserved
PROCEDURAL	Non-reserved	N/A	N/A
PROCEDURE	Reserved	Reserved	Reserved
PROFILE	Non-reserved	N/A	N/A
PUBLIC	N/A	Reserved	Reserved
PUBLICATION	Non-reserved	N/A	N/A
PUBLISH	Non-reserved	N/A	N/A
PURGE	Non-reserved	N/A	N/A
QUERY	Non-reserved	N/A	N/A
QUOTE	Non-reserved	N/A	N/A
RANDOMIZED	Non-reserved	N/A	N/A
RANGE	Non-reserved	N/A	N/A
RATIO	Non-reserved	N/A	N/A
RAW	Non-reserved	N/A	N/A
READ	Non-reserved	Reserved	Reserved
READS	N/A	Reserved	N/A
REAL	Non-reserved (excluding functions and types)	Reserved	Reserved
REASSIGN	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
REBUILD	Non-reserved	N/A	N/A
RECHECK	Non-reserved	N/A	N/A
RECURSIVE	Non-reserved	Reserved	N/A
RECYCLEBIN	Reserved (functions and types allowed)	N/A	N/A
REDISANYVALUE	Non-reserved	N/A	N/A
REF	Non-reserved	Reserved	N/A
REFERENCES	Reserved	Reserved	Reserved
REFERENCING	N/A	Reserved	N/A
REFRESH	Non-reserved	N/A	N/A
REINDEX	Non-reserved	N/A	N/A
REJECT	Reserved	N/A	N/A
RELATIVE	Non-reserved	Reserved	Reserved
RELEASE	Non-reserved	N/A	N/A
RELOPTIONS	Non-reserved	N/A	N/A
REMOTE	Non-reserved	N/A	N/A
REMOVE	Non-reserved	N/A	N/A
RENAME	Non-reserved	N/A	N/A
REPEATABLE	Non-reserved	Non-reserved	Non-reserved
REPLACE	Non-reserved	N/A	N/A
REPLICA	Non-reserved	N/A	N/A
RESET	Non-reserved	N/A	N/A
RESIZE	Non-reserved	N/A	N/A
RESOURCE	Non-reserved	N/A	N/A
RESTART	Non-reserved	N/A	N/A
RESTRICT	Non-reserved	Reserved	Reserved
RESULT	N/A	Reserved	N/A
RETURN	Non-reserved	Reserved	N/A
RETURNED_LENGTH	N/A	Non-reserved	Non-reserved

Keyword	GaussDB	SQL:1999	SQL-92
RETURNED_OCTET_LENGTH	N/A	Non-reserved	Non-reserved
RETURNED_SQLSTATE	N/A	Non-reserved	Non-reserved
RETURNING	Reserved	N/A	N/A
RETURNS	Non-reserved	Reserved	N/A
REUSE	Non-reserved	N/A	N/A
REVOKE	Non-reserved	Reserved	Reserved
RIGHT	Reserved (functions and types allowed)	Reserved	Reserved
ROLE	Non-reserved	Reserved	N/A
ROLES	Non-reserved	N/A	N/A
ROLLBACK	Non-reserved	Reserved	Reserved
ROLLUP	Non-reserved	Reserved	N/A
ROTATION	Non-reserved	N/A	N/A
ROUTINE	N/A	Reserved	N/A
ROUTINE_CATALOG	N/A	Non-reserved	N/A
ROUTINE_NAME	N/A	Non-reserved	N/A
ROUTINE_SCHEMA	N/A	Non-reserved	N/A
ROW	Non-reserved (excluding functions and types)	Reserved	N/A
ROWS	Non-reserved	Reserved	Reserved
ROWTYPE	Non-reserved	N/A	N/A
ROW_COUNT	N/A	Non-reserved	Non-reserved
RULE	Non-reserved	N/A	N/A
ROWNUM	Reserved	Reserved	N/A
SAMPLE	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
SAVEPOINT	Non-reserved	Reserved	N/A
SCALE	N/A	Non-reserved	Non-reserved
SCHEMA	Non-reserved	Reserved	Reserved
SCHEMA_NAME	N/A	Non-reserved	Non-reserved
SCOPE	N/A	Reserved	N/A
SCROLL	Non-reserved	Reserved	Reserved
SEARCH	Non-reserved	Reserved	N/A
SECOND	Non-reserved	Reserved	Reserved
SECTION	N/A	Reserved	Reserved
SECURITY	Non-reserved	Non-reserved	N/A
SELECT	Reserved	Reserved	Reserved
SELF	N/A	Non-reserved	N/A
SENSITIVE	N/A	Non-reserved	N/A
SEQUENCE	Non-reserved	Reserved	N/A
SEQUENCES	Non-reserved	N/A	N/A
SERIALIZABLE	Non-reserved	Non-reserved	Non-reserved
SERVER	Non-reserved	N/A	N/A
SERVER_NAME	N/A	Non-reserved	Non-reserved
SESSION	Non-reserved	Reserved	Reserved
SESSION_USER	Reserved	Reserved	Reserved
SET	Non-reserved	Reserved	Reserved
SETOF	Non-reserved (excluding functions and types)	N/A	N/A
SETS	Non-reserved	Reserved	N/A
SHARE	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
SHIPPABLE	Non-reserved	N/A	N/A
SHOW	Non-reserved	N/A	N/A
SHUTDOWN	Non-reserved	N/A	N/A
SIBLINGS	Non-reserved	N/A	N/A
SIMILAR	Reserved (functions and types allowed)	Non-reserved	N/A
SIMPLE	Non-reserved	Non-reserved	N/A
SIZE	Non-reserved	Reserved	Reserved
SKIP	Non-reserved	N/A	N/A
SLICE	Non-reserved	N/A	N/A
SMALLDATETIME_FORMAT	Non-reserved	N/A	N/A
SMALLDATETIME	Non-reserved (excluding functions and types)	N/A	N/A
SMALLINT	Non-reserved (excluding functions and types)	Reserved	Reserved
SNAPSHOT	Non-reserved	N/A	N/A
SOME	Reserved	Reserved	Reserved
SOURCE	Non-reserved	Non-reserved	N/A
SPACE	Non-reserved	Reserved	Reserved
SPECIFIC	N/A	Reserved	N/A
SPECIFICTYPE	N/A	Reserved	N/A
SPECIFIC_NAME	N/A	Non-reserved	N/A
SPILL	Non-reserved	N/A	N/A
SPLIT	Non-reserved	N/A	N/A
SQL	N/A	Reserved	Reserved
SQLCODE	N/A	N/A	Reserved

Keyword	GaussDB	SQL:1999	SQL-92
SQLERROR	N/A	N/A	Reserved
SQLEXCEPTION	N/A	Reserved	N/A
SQLSTATE	N/A	Reserved	Reserved
SQLWARNING	N/A	Reserved	N/A
STABLE	Non-reserved	N/A	N/A
STANDALONE	Non-reserved	N/A	N/A
START	Non-reserved	Reserved	N/A
STATE	N/A	Reserved	N/A
STATEMENT	Non-reserved	Reserved	N/A
STATEMENT_ID	Non-reserved	N/A	N/A
STATIC	N/A	Reserved	N/A
STATISTICS	Non-reserved	N/A	N/A
STDIN	Non-reserved	N/A	N/A
STDOUT	Non-reserved	N/A	N/A
STORAGE	Non-reserved	N/A	N/A
STORE	Non-reserved	N/A	N/A
STORED	Non-reserved	N/A	N/A
STRATIFY	Non-reserved	N/A	N/A
STREAM	Non-reserved	N/A	N/A
STRICT	Non-reserved	N/A	N/A
STRIP	Non-reserved	N/A	N/A
STRUCTURE	N/A	Reserved	N/A
STYLE	N/A	Non-reserved	N/A
SUBCLASS_ORIGIN	N/A	Non-reserved	Non-reserved
SUBLIST	N/A	Non-reserved	N/A
SUBPARTITION	Non-reserved	N/A	N/A
SUBSCRIPTION	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
SUBSTRING	Non-reserved (excluding functions and types)	Non-reserved	Reserved
SUM	N/A	Non-reserved	Reserved
SYMMETRIC	Reserved	Non-reserved	N/A
SYNONYM	Non-reserved	N/A	N/A
SYS_REFCURSOR	Non-reserved	N/A	N/A
SYSDATE	Reserved	N/A	N/A
SYSID	Non-reserved	N/A	N/A
SYSTEM	Non-reserved	Non-reserved	N/A
SYSTEM_USER	N/A	Reserved	Reserved
TABLE	Reserved	Reserved	Reserved
TABLES	Non-reserved	N/A	N/A
TABLESAMPLE	Reserved (functions and types allowed)	N/A	N/A
TABLESPACE	Non-reserved	N/A	N/A
TABLE_NAME	N/A	Non-reserved	Non-reserved
TARGET	Non-reserved	N/A	N/A
TIME_FORMAT	Non-reserved	N/A	N/A
TIMESTAMP_FORMAT	Non-reserved	N/A	N/A
TEMP	Non-reserved	N/A	N/A
TEMPLATE	Non-reserved	N/A	N/A
TEMPORARY	Non-reserved	Reserved	Reserved
TERMINATE	N/A	Reserved	N/A
TERMINATED	Non-reserved	N/A	N/A
TEXT	Non-reserved	N/A	N/A
THAN	Non-reserved	Reserved	N/A
THEN	Reserved	Reserved	Reserved

Keyword	GaussDB	SQL:1999	SQL-92
TIME	Non-reserved (excluding functions and types)	Reserved	Reserved
TIMECAPSULE	Reserved (functions and types allowed)	N/A	N/A
TIMESTAMP	Non-reserved (excluding functions and types)	Reserved	Reserved
TIMESTAMPDIFF	Non-reserved (excluding functions and types)	N/A	N/A
TIMEZONE_HOUR	N/A	Reserved	Reserved
TIMEZONE_MINUTE	N/A	Reserved	Reserved
TINYINT	Non-reserved (excluding functions and types)	N/A	N/A
TO	Reserved	Reserved	Reserved
TRAILING	Reserved	Reserved	Reserved
TRANSACTION	Non-reserved	Reserved	Reserved
TRANSACTIONS_COMMITTED	N/A	Non-reserved	N/A
TRANSACTIONS_ROLLED_BACK	N/A	Non-reserved	N/A
TRANSACTION_ACTIVE	N/A	Non-reserved	N/A
TRANSFORM	Non-reserved	Non-reserved	N/A
TRANSFORMS	N/A	Non-reserved	N/A
TRANSLATE	N/A	Non-reserved	Reserved
TRANSLATION	N/A	Reserved	Reserved

Keyword	GaussDB	SQL:1999	SQL-92
TREAT	Non-reserved (excluding functions and types)	Reserved	N/A
TRIGGER	Non-reserved	Reserved	N/A
TRIGGER_CATALOG	N/A	Non-reserved	N/A
TRIGGER_NAME	N/A	Non-reserved	N/A
TRIGGER_SCHEMA	N/A	Non-reserved	N/A
TRIM	Non-reserved (excluding functions and types)	Non-reserved	Reserved
TRUE	Reserved	Reserved	Reserved
TRUNCATE	Non-reserved	N/A	N/A
TRUSTED	Non-reserved	N/A	N/A
TSFIELD	Non-reserved	N/A	N/A
TSTAG	Non-reserved	N/A	N/A
TSTIME	Non-reserved	N/A	N/A
TYPE	Non-reserved	Non-reserved	Non-reserved
TYPES	Non-reserved	N/A	N/A
UESCAPE	N/A	N/A	N/A
UNBOUNDED	Non-reserved	N/A	N/A
UNCOMMITTED	Non-reserved	Non-reserved	Non-reserved
UNDER	N/A	Reserved	N/A
UNENCRYPTED	Non-reserved	N/A	N/A
UNION	Reserved	Reserved	Reserved
UNIQUE	Reserved	Reserved	Reserved
UNKNOWN	Non-reserved	Reserved	Reserved
UNLIMITED	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
UNLISTEN	Non-reserved	N/A	N/A
UNLOCK	Non-reserved	N/A	N/A
UNLOGGED	Non-reserved	N/A	N/A
UNNAMED	N/A	Non-reserved	Non-reserved
UNNEST	N/A	Reserved	N/A
UNTIL	Non-reserved	N/A	N/A
UNUSABLE	Non-reserved	N/A	N/A
UPDATE	Non-reserved	Reserved	Reserved
UPPER	N/A	Non-reserved	Reserved
USAGE	N/A	Reserved	Reserved
USEEOF	Non-reserved	N/A	N/A
USER	Reserved	Reserved	Reserved
USER_DEFINED_TYPE_CATALOG	N/A	Non-reserved	N/A
USER_DEFINED_TYPE_NAME	N/A	Non-reserved	N/A
USER_DEFINED_TYPE_SCHEMA	N/A	Non-reserved	N/A
USING	Reserved	Reserved	Reserved
VACUUM	Non-reserved	N/A	N/A
VALID	Non-reserved	N/A	N/A
VALIDATE	Non-reserved	N/A	N/A
VALIDATION	Non-reserved	N/A	N/A
VALIDATOR	Non-reserved	N/A	N/A
VALUE	Non-reserved	Reserved	Reserved
VALUES	Non-reserved (excluding functions and types)	Reserved	Reserved

Keyword	GaussDB	SQL:1999	SQL-92
VARCHAR	Non-reserved (excluding functions and types)	Reserved	Reserved
VARCHAR2	Non-reserved (excluding functions and types)	N/A	N/A
VARIABLE	N/A	Reserved	N/A
VARIABLES	Non-reserved	N/A	N/A
VARIADIC	Reserved	N/A	N/A
VARYING	Non-reserved	Reserved	Reserved
VCGROUP	Non-reserved	N/A	N/A
VERBOSE	Reserved (functions and types allowed)	N/A	N/A
VERSION	Non-reserved	N/A	N/A
VERIFY	Reserved	N/A	N/A
VIEW	Non-reserved	Reserved	Reserved
VOLATILE	Non-reserved	N/A	N/A
WAIT	Non-reserved	N/A	N/A
WEAK	Non-reserved	N/A	N/A
WHEN	Reserved	Reserved	Reserved
WHENEVER	N/A	Reserved	Reserved
WHERE	Reserved	Reserved	Reserved
WHITESPACE	Non-reserved	N/A	N/A
WINDOW	Reserved	N/A	N/A
WITH	Reserved	Reserved	Reserved
WITHIN	Non-reserved	N/A	N/A
WITHOUT	Non-reserved	Reserved	N/A
WORK	Non-reserved	Reserved	Reserved
WORKLOAD	Non-reserved	N/A	N/A
WRAPPER	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
WRITE	Non-reserved	Reserved	Reserved
XML	Non-reserved	N/A	N/A
XMLATTRIBUTES	Non-reserved (excluding functions and types)	N/A	N/A
XMLCONCAT	Non-reserved (excluding functions and types)	N/A	N/A
XMLELEMENT	Non-reserved (excluding functions and types)	N/A	N/A
XMLEXISTS	Non-reserved (excluding functions and types)	N/A	N/A
XMLFOREST	Non-reserved (excluding functions and types)	N/A	N/A
XMLPARSE	Non-reserved (excluding functions and types)	N/A	N/A
XMLPI	Non-reserved (excluding functions and types)	N/A	N/A
XMLROOT	Non-reserved (excluding functions and types)	N/A	N/A
XMLSERIALIZE	Non-reserved (excluding functions and types)	N/A	N/A
YEAR	Non-reserved	Reserved	Reserved
YES	Non-reserved	N/A	N/A
ZONE	Non-reserved	Reserved	Reserved

Fields listed in the following table cannot be used as column names during table creation.

CTID	XMIN	CMIN	XMAX	CMAX
TABLEOID	XC_NODE_ID	TID	OID	GS_TUPLE_UID
TABLEBUCKETID	-	N/A	N/A	-

11.3 Data Types

GaussDB supports implicit conversions between certain data types. For details, see**PG_CAST** .

11.3.1 Numeric Types

Table 11-2 lists all available types. For digit operators and related built-in functions, see **Mathematical Functions and Operators**.

Table 11-2 Integer types

Name	Description	Storage Space	Range
TINYINT	Tiny integer, also called INT1	1 byte	0-255
SMALLINT	Small integer, also called INT2	2 bytes	–32,768 to +32,767
INTEGER	Typical choice for integers, also called INT4	4 bytes	–2,147,483,648 to +2,147,483,647
BINARY_INTEGER	Alias of INTEGER.	4 bytes	–2,147,483,648 to +2,147,483,647
BIGINT	Big integer, also called INT8	8 bytes	–9,223,372,036,854,775,808 to +9,223,372,036,854,775,807
int16	A 16-byte integer cannot be used to create tables.	16 bytes	–170,141,183,460,469,231,731,687,303,715,884,105,728 to +170,141,183,460,469,231,731,687,303,715,884,105,727

Example:

```
-- Create a table containing TINYINT data.
openGauss=# CREATE TABLE int_type_t1
           (
            IT_COL1 TINYINT
           );

-- Insert data to the created table.
openGauss=# INSERT INTO int_type_t1 VALUES(10);

-- View data.
openGauss=# SELECT * FROM int_type_t1;
 it_col1
---------
 10
(1 row)

-- Delete the table.
openGauss=# DROP TABLE int_type_t1;
-- Create a table containing TINYINT, INTEGER, and BIGINT data.
openGauss=# CREATE TABLE int_type_t2
(
    a TINYINT,
    b TINYINT,
    c INTEGER,
    d BIGINT
);

-- Insert data.
openGauss=# INSERT INTO int_type_t2 VALUES(100, 10, 1000, 10000);

-- View data.
openGauss=# SELECT * FROM int_type_t2;
  a  | b  |  c   |   d
-----+----+------+-------
 100 | 10 | 1000 | 10000
(1 row)

-- Delete the table.
openGauss=# DROP TABLE int_type_t2;
```

NOTE

- Only numbers of the TINYINT, SMALLINT, INTEGER, BIGINT, or INT16 type, that is, integers can be stored. Saving a number with a decimal in any of the data types will result in errors.
- The INTEGER type is the common choice, as it offers the best balance between range, storage size, and performance. Generally, use the SMALLINT type only if you are sure that the value range is within the SMALLINT value range. The storage speed of INTEGER is much faster. BIGINT is used only when the range of INTEGER is not large enough.

Table 11-3 Arbitrary precision types

Name	Description	Storage Space	Range
NUMERIC[(p[,s])], DECIMAL[(p[,s])]	The value range of **p** (precision) is [1,1000], and the value range of **s** (scale) is [0,*p*]. **NOTE** **p** indicates the total digits, and **s** indicates the decimal digit.	The precision is specified by users. Every four decimal digits occupy two bytes, and an extra eight-byte overhead is added to the entire data.	Up to 131,072 digits before the decimal point; and up to 16,383 digits after the decimal point when no precision is specified
NUMBER[(p[,s])]	Alias of the NUMERIC type.	The precision is specified by users. Every four decimal digits occupy two bytes, and an extra eight-byte overhead is added to the entire data.	Up to 131,072 digits before the decimal point; and up to 16,383 digits after the decimal point when no precision is specified

Example:

```
-- Create a table.
openGauss=# CREATE TABLE decimal_type_t1
(
    DT_COL1 DECIMAL(10,4)
);

-- Insert data.
openGauss=# INSERT INTO decimal_type_t1 VALUES(123456.122331);

-- Query data in the table.
openGauss=# SELECT * FROM decimal_type_t1;
   dt_col1
-------------
 123456.1223
(1 row)

-- Delete the table.
openGauss=# DROP TABLE decimal_type_t1;
-- Create a table.
openGauss=# CREATE TABLE numeric_type_t1
(
    NT_COL1 NUMERIC(10,4)
);

-- Insert data.
openGauss=# INSERT INTO numeric_type_t1 VALUES(123456.12354);

-- Query data in the table.
openGauss=# SELECT * FROM numeric_type_t1;
   nt_col1
-------------
 123456.1235
(1 row)

-- Delete the table.
openGauss=# DROP TABLE numeric_type_t1;
```

NOTE

- Compared to the integer types, the arbitrary precision numbers require larger storage space and have lower storage efficiency, operation efficiency, and poorer compression ratio results. The INTEGER type is the common choice when number types are defined. Arbitrary precision numbers are used only when numbers exceed the maximum range indicated by the integers.
- When NUMERIC/DECIMAL is used for defining a column, you are advised to specify the precision (p) and scale (s) for the column.

Table 11-4 Sequence integer

Name	Description	Storage Space	Range
SMALLSERIAL	Two-byte auto-incrementing integer	2 bytes	–32,768 to +32,767
SERIAL	Four-byte auto-incrementing integer	4 bytes	–2,147,483,648 to +2,147,483,647
BIGSERIAL	Eight-byte auto-incrementing integer	8 bytes	–9,223,372,036,854,775,808 to +9,223,372,036,854,775,807
LARGESERIAL	By default, a 16-byte auto-incrementing integer is inserted. The actual value type is the same as that of numeric.	Variable-length type. Every four decimal digits occupy two bytes, and an extra eight-byte overhead is added to the entire data.	There can be a maximum of 131072 digits before the decimal point and 16383 digits after the decimal point.

Example:

```
-- Create a table.
openGauss=# CREATE TABLE smallserial_type_tab(a SMALLSERIAL);

-- Insert data.
openGauss=# INSERT INTO smallserial_type_tab VALUES(default);
```

```
-- Insert data again.
openGauss=# INSERT INTO smallserial_type_tab VALUES(default);

-- View data.
openGauss=# SELECT * FROM smallserial_type_tab;
 a
---
 1
 2
(2 rows)

-- Create a table.
openGauss=# CREATE TABLE serial_type_tab(b SERIAL);

-- Insert data.
openGauss=# INSERT INTO serial_type_tab VALUES(default);

-- Insert data again.
openGauss=# INSERT INTO serial_type_tab VALUES(default);

-- View data.
openGauss=# SELECT * FROM serial_type_tab;
 b
---
 1
 2
(2 rows)

-- Create a table.
openGauss=# CREATE TABLE bigserial_type_tab(c BIGSERIAL);

-- Insert data.
openGauss=# INSERT INTO bigserial_type_tab VALUES(default);

-- Insert data.
openGauss=# INSERT INTO bigserial_type_tab VALUES(default);

-- View data.
openGauss=# SELECT * FROM bigserial_type_tab;
 c
---
 1
 2
(2 rows)

-- Create a table.
openGauss=# CREATE TABLE largeserial_type_tab(c LARGESERIAL);

-- Insert data.
openGauss=# INSERT INTO largeserial_type_tab VALUES(default);

-- Insert data.
openGauss=# INSERT INTO largeserial_type_tab VALUES(default);

-- View data.
openGauss=# SELECT * FROM largeserial_type_tab;
 c
---
 1
 2
(2 rows)

-- Delete the table.
openGauss=# DROP TABLE smallserial_type_tab;

openGauss=# DROP TABLE serial_type_tab;

openGauss=# DROP TABLE bigserial_type_tab;
```

NOTE

SMALLSERIAL, SERIAL, BIGSERIAL, and LARGESERIAL are not real types. They are concepts used for setting a unique identifier for a table. Therefore, an integer column is created and its default value plans to be read from a sequencer. A NOT NULL constraint is used to ensure NULL is not inserted. In most cases you would also want to attach a **UNIQUE** or **PRIMARY KEY** constraint to prevent duplicate values from being inserted unexpectedly, but this is not automatic. The sequencer is set so that it belongs to the column. In this case, when the column or the table is deleted, the sequencer is also deleted. Currently, the **SERIAL** column can be specified only when you create a table. You cannot add the **SERIAL** column in an existing table. In addition, **SERIAL** columns cannot be created in temporary tables. Because SERIAL is not a data type, columns cannot be converted to this type.

Table 11-5 Floating point types

Name	Description	Storage Space	Range
REAL, FLOAT4	Single precision floating points, inexact	4 bytes	–3.402E+38 to 3.402E +38, 6-bit decimal digits
DOUBLE PRECISION, FLOAT8	Double precision floating points, inexact	8 bytes	–1.79E+308 to 1.79E +308, 15-bit decimal digits
FLOAT[(p)]	Floating points, inexact. The value range of **p** (precision) is [1,53]. **NOTE** **p** is the precision, indicating the total number of binary bits.	4 or 8 bytes	**REAL** or **DOUBLE PRECISION** is selected as an internal identifier based on precision (p). If no precision is specified, DOUBLE PRECISION is used as the internal identifier.
BINARY_DOUBLE	Alias of DOUBLE PRECISION.	8 bytes	–1.79E+308 to 1.79E +308, 15-bit decimal digits
DEC[(p[,s])]	The value range of **p** (precision) is [1,1000], and the value range of **s** (scale) is [0,*p*]. **NOTE** **p** indicates the total digits, and **s** indicates the decimal digit.	The precision is specified by users. Every four decimal digits occupy two bytes, and an extra eight-byte overhead is added to the entire data.	Up to 131,072 digits before the decimal point; and up to 16,383 digits after the decimal point when no precision is specified

Name	Description	Storage Space	Range
INTEGER[(p[,s])]	The value range of **p** (precision) is [1,1000], and the value range of **s** (scale) is [0,*p*].	The precision is specified by users. Every four decimal digits occupy two bytes, and an extra eight-byte overhead is added to the entire data.	N/A

Example:

```
-- Create a table.
openGauss=# CREATE TABLE float_type_t2
(
    FT_COL1 INTEGER,
    FT_COL2 FLOAT4,
    FT_COL3 FLOAT8,
    FT_COL4 FLOAT(3),
    FT_COL5 BINARY_DOUBLE,
    FT_COL6 DECIMAL(10,4),
    FT_COL7 INTEGER(6,3)
);

-- Insert data.
openGauss=# INSERT INTO float_type_t2 VALUES(10,10.365456,123456.1234,10.3214, 321.321, 123.123654, 123.123654);

-- View data.
openGauss=# SELECT * FROM float_type_t2 ;
 ft_col1 | ft_col2 |   ft_col3   | ft_col4 | ft_col5 | ft_col6  | ft_col7
---------+---------+-------------+---------+---------+----------+---------
      10 | 10.3655 | 123456.1234 | 10.3214 | 321.321 | 123.1237 | 123.124
(1 row)

-- Delete the table.
openGauss=# DROP TABLE float_type_t2;
```

11.3.2 Monetary Types

The money type stores a currency amount with fixed fractional precision.

The range shown in **Table 11-6** assumes there are two fractional digits. Input is accepted in a variety of formats, including integer and floating-point literals, as well as typical currency formatting, such as "$1,000.00". Output is generally in the last format but depends on the locale.

Table 11-6 Monetary type

Name	Storage Space	Description	Range
money	8 bytes	Currency amount	–92233720368547758.08 to +92233720368547758.07

Values of the numeric, int, and bigint data types can be cast to money. Conversion from the real and double precision data types can be done by casting to numeric first, for example:

```
openGauss=# SELECT '12.34'::float8::numeric::money;
```

However, this is not recommended. Floating point numbers should not be used to handle money due to the potential for rounding errors.

A money value can be cast to numeric without loss of precision. Conversion to other types could potentially lose precision, and must also be done in two stages:

```
openGauss=# SELECT '52093.89'::money::numeric::float8;
```

When a money value is divided by another money value, the result is of the double precision type (that is, a pure number, not money); the currency units cancel each other out in the division.

11.3.3 Boolean Types

Table 11-7 Boolean types

Name	Description	Storage Space	Value
BOOLEAN	Boolean type	1 byte	● **true** ● **false** ● **null** (unknown)

- Valid literal values for the "true" state include:

  TRUE, **'t'**, **'true'**, **'y'**, **'yes'**, **'1'**, **'TRUE'**, **true**, and an integer ranging from 1 to 2^63 – 1 or from –1 to –2^63.
- Valid literal values for the "false" state include:

  FALSE, **'f'**, **'false'**, **'n'**, **'no'**, **'0'**, **0**, **'FALSE'**, and **false**.

TRUE and **FALSE** are standard expressions, compatible with SQL statements.

Examples

Boolean values are displayed using the letters t and f.

```
-- Create a table.
openGauss=# CREATE TABLE bool_type_t1
(
    BT_COL1 BOOLEAN,
    BT_COL2 TEXT
);

-- Insert data.
openGauss=# INSERT INTO bool_type_t1 VALUES (TRUE, 'sic est');

openGauss=# INSERT INTO bool_type_t1 VALUES (FALSE, 'non est');

-- View data.
openGauss=# SELECT * FROM bool_type_t1;
 bt_col1 | bt_col2
```

```
---------+---------
 t       | sic est
 f       | non est
(2 rows)

openGauss=# SELECT * FROM bool_type_t1 WHERE bt_col1 = 't';
 bt_col1 | bt_col2
---------+---------
 t       | sic est
(1 row)

-- Delete the table.
openGauss=# DROP TABLE bool_type_t1;
```

11.3.4 Character Types

GaussDB lists the character data types supported by **Table 11-8**. For string operators and related built-in functions, see **Character Processing Functions and Operators**.

Table 11-8 Character types

Name	Description	Storage Space

CHAR(n)
CHARACTER(n)
NCHAR(n)
	Fixed-length character string. Empty characters are filled in with blank spaces. **n** indicates the string length. If it is not specified, the default precision **1** is used.	The maximum size is 10 MB.

VARCHAR(n) CHARACTER VARYING(n)	Variable-length string. In PostgreSQL-compatible mode, **n** indicates the string length. In other compatibility modes, **n** indicates the byte length.	The maximum size is 10 MB.
VARCHAR2(n)	Variable-length string. It is the alias of the VARCHAR(n) type. **n** indicates the string length.	The maximum size is 10 MB.
NVARCHAR2(n)	Variable-length string. **n** indicates the string length.	The maximum size is 10 MB.

Name	Description	Storage Space
TEXT	Variable-length string.	The maximum size is 1 GB minus 1 byte. However, the size of the column description header and the size of the tuple (less than 1 GB minus 1 byte) where the column is located must also be considered. Therefore, the maximum size of the TEXT type may be less than 1 GB minus 1 byte.
CLOB	Big text object. It is the alias of the TEXT type.	The maximum size is 32 TB minus 1 byte. However, the size of the column description header and the size of the tuple (less than 32 TB minus 1 byte) where the column is located must also be considered. Therefore, the maximum size of the CLOB type may be less than 32 TB minus 1 byte.

NOTE

1. In addition to the restriction on the size of each column, the total size of each tuple cannot exceed 1 GB minus 1 byte and is affected by the control header information of the column, the control header information of the tuple, and whether null fields exist in the tuple.
2. NCHAR is the alias of the bpchar type, and VARCHAR2(n) is the alias of the VARCHAR(n) type.
3. Only advanced package dbe_lob supports CLOBs whose size is greater than 1 GB. System functions do not support CLOBs whose size is greater than 1 GB.

In GaussDB, there are two other fixed-length character types, as shown in **Table 11-9**. The **name** type exists only for the storage of identifiers in the internal system catalogs and is not intended for use by general users. Its length is currently defined as 64 bytes (63 usable characters plus terminator). The type **"char"** only uses one byte of storage. It is internally used in the system catalogs as a simplistic enumeration type.

Table 11-9 Special character types

Name	Description	Storage Space
name	Internal type for object names	64 bytes
"char"	Single-byte internal type	1 byte

Examples

```
-- Create a table.
openGauss=# CREATE TABLE char_type_t1
(
    CT_COL1 CHARACTER(4)
);

-- Insert data.
openGauss=# INSERT INTO char_type_t1 VALUES ('ok');

-- Query data in the table.
openGauss=# SELECT ct_col1, char_length(ct_col1) FROM char_type_t1;
 ct_col1 | char_length
---------+-------------
 ok      |           4
(1 row)

-- Delete the table.
openGauss=# DROP TABLE char_type_t1;

-- Create a table.
openGauss=# CREATE TABLE char_type_t2
(
    CT_COL1 VARCHAR(5)
);

-- Insert data.
openGauss=# INSERT INTO char_type_t2 VALUES ('ok');

openGauss=# INSERT INTO char_type_t2 VALUES ('good');

-- Specify the type length. An error is reported if an inserted string exceeds this length.
openGauss=# INSERT INTO char_type_t2 VALUES ('too long');
ERROR:  value too long for type character varying(5)
CONTEXT:  referenced column: ct_col1

-- Specify the type length. A string exceeding this length is truncated.
openGauss=# INSERT INTO char_type_t2 VALUES ('too long'::varchar(5));

-- Query data.
openGauss=# SELECT ct_col1, char_length(ct_col1) FROM char_type_t2;
 ct_col1 | char_length
---------+-------------
 ok      |           2
 good    |           4
 too l   |           5
(3 rows)

-- Delete data.
openGauss=# DROP TABLE char_type_t2;
```

11.3.5 Binary Types

Table 11-10 lists the binary data types supported by GaussDB.

Table 11-10 Binary types

Name	Description	Storage Space
BLOB	Binary large object. Currently, BLOB only supports the following external access APIs: ● DBE_LOB.GET_LENGTH ● DBE_LOB.READ ● DBE_LOB.WRITE ● DBE_LOB.WRITE_APPEND ● DBE_LOB.COPY ● DBE_LOB.ERASE For details about the APIs, see DBE_LOB. **NOTE** Column storage cannot be used for the BLOB type.	The maximum size is 32 TB (that is, 35184372088832 bytes).
RAW	Variable-length hexadecimal string. **NOTE** Column store cannot be used for the raw type.	4 bytes plus the actual hexadecimal string. Its maximum length is 1073733621 bytes (1 GB – 8203 bytes).
BYTEA	Variable-length binary string.	4 bytes plus the actual binary string. Its maximum length is 1073733621 bytes (1 GB – 8203 bytes).
BYTEAWITHOUTORDERWITHEQUALCOL	Variable-length binary character string (new type for the encryption feature. If the encryption type of the encrypted column is specified as deterministic encryption, the column type is BYTEAWITHOUTORDER-WITHEQUALCOL). The original data type is displayed when the encrypted table is printed by running the meta command.	4 bytes plus the actual binary string. The maximum value is 1,073,741,771 bytes (1 GB minus 53 bytes).

Name	Description	Storage Space
BYTEAWITHOUTORDERCOL	Variable-length binary character string (new type for the encryption feature. If the encryption type of the encrypted column is specified as random encryption, the column type is BYTEAWITHOUTORDER-COL). The original data type is displayed when the encrypted table is printed by running the meta command.	4 bytes plus the actual binary string. The maximum value is 1,073,741,771 bytes (1 GB minus 53 bytes).
_BYTEAWITHOUTORDERWITHEQUALCOL	Variable-length binary string, which is a new type for the encryption feature.	4 bytes plus the actual binary string. The maximum value is 1,073,741,771 bytes (1 GB minus 53 bytes).
_BYTEAWITHOUTORDERCOL	Variable-length binary string, which is a new type for the encryption feature.	4 bytes plus the actual binary string. The maximum value is 1,073,741,771 bytes (1 GB minus 53 bytes).

NOTE

- In addition to the size limitation on each column, the total size of each tuple is 1073733621 bytes (1 GB to 8203 bytes).
- BYTEAWITHOUTORDERWITHEQUALCOL, BYTEAWITHOUTORDERCOL, _BYTEAWITHOUTORDERWITHEQUALCOL, and _BYTEAWITHOUTORDERCOL cannot be directly used to create a table.

Example:

```
-- Create a table.
openGauss=# CREATE TABLE blob_type_t1
(
    BT_COL1 INTEGER,
    BT_COL2 BLOB,
    BT_COL3 RAW,
    BT_COL4 BYTEA
) ;

-- Insert data.
openGauss=# INSERT INTO blob_type_t1 VALUES(10,empty_blob(),
HEXTORAW('DEADBEEF'),E'\\xDEADBEEF');

-- Query data in the table.
openGauss=# SELECT * FROM blob_type_t1;
 bt_col1 | bt_col2 | bt_col3  |  bt_col4
---------+---------+----------+------------
      10 |         | DEADBEEF | \xdeadbeef
(1 row)

-- Delete the table.
openGauss=# DROP TABLE blob_type_t1;
```

11.3.6 Date/Time Types

Table 11-11 lists the date/time types supported by GaussDB. For the operators and built-in functions of the types, see **Date and Time Processing Functions and Operators**.

NOTE

If the time format of another database is different from that of GaussDB, modify the value of the **DateStyle** parameter to keep them consistent.

Table 11-11 Date/Time types

Name	Description	Storage Space
DATE	Date. **NOTE** For A compatibility, the database treats empty strings as **NULL** and replaces DATE with TIMESTAMP(0) WITHOUT TIME ZONE.	4 bytes (8 bytes in A compatibility schema)
TIME [(p)] [WITHOUT TIME ZONE]	Time within one day. **p** indicates the precision after the decimal point. The value ranges from 0 to 6.	8 bytes
TIME [(p)] [WITH TIME ZONE]	Time within one day (with time zone). **p** indicates the precision after the decimal point. The value ranges from 0 to 6.	12 bytes
TIMESTAMP[(p)] [WITHOUT TIME ZONE]	Date and time. **p** indicates the precision after the decimal point. The value ranges from 0 to 6.	8 bytes
TIMESTAMP[(p)] [WITH TIME ZONE]	Date and time (with time zone). TIMESTAMP is also called TIMESTAMPTZ. **p** indicates the precision after the decimal point. The value ranges from 0 to 6.	8 bytes
SMALLDATETIME	Date and time (without time zone). The precision is minute. A duration between 30s and 60s is rounded into 1 minute.	8 bytes

Name	Description	Storage Space
INTERVAL DAY (l) TO SECOND (p)	Time interval (X days X hours X minutes X seconds). ● **l**: indicates the precision of days. The value ranges from 0 to 6. For compatibility, the precision functions are not supported. ● **p**: indicates the precision of seconds. The value ranges from 0 to 6. The digit 0 at the end of a decimal number is not displayed.	16 bytes
INTERVAL [FIELDS] [(p)]	Time interval. ● **FIELDS**: **YEAR**, **MONTH**, **DAY**, **HOUR**, **MINUTE**, **SECOND**, **DAY TO HOUR**, **DAY TO MINUTE**, **DAY TO SECOND**, **HOUR TO MINUTE**, **HOUR TO SECOND**, or **MINUTE TO SECOND**. ● **p**: indicates the precision of seconds. The value ranges from 0 to 6. **p** takes effect only when **FIELDS** is **SECOND**, **DAY TO SECOND**, **HOUR TO SECOND**, or **MINUTE TO SECOND**. The digit 0 at the end of a decimal number is not displayed.	12 bytes
reltime	Relative time interval. The format is as follows: *X* years *X* months *X* days *XX:XX:XX* The Julian calendar is used. It specifies that a year has 365.25 days and a month has 30 days. The relative time interval needs to be calculated based on the input value. The output format is POSTGRES.	4 bytes
abstime	Date and time. The format is as follows: YYYY-MM-DD hh:mm:ss+timezone The value range is from 1901-12-13 20:45:53 GMT to 2038-01-18 23:59:59 GMT. The precision is second.	4 bytes

Example:

```
-- Create a table.
openGauss=# CREATE TABLE date_type_tab(coll date);

-- Insert data.
openGauss=# INSERT INTO date_type_tab VALUES (date '12-10-2010');

-- View data.
openGauss=# SELECT * FROM date_type_tab;
       coll
---------------------
 2010-12-10(1 row)

-- Delete the table.
openGauss=# DROP TABLE date_type_tab;

-- Create a table.
openGauss=# CREATE TABLE time_type_tab (da time without time zone ,dai time with time zone,dfgh
timestamp without time zone,dfga timestamp with time zone, vbg smalldatetime);

-- Insert data.
openGauss=# INSERT INTO time_type_tab VALUES ('21:21:21','21:21:21 pst','2010-12-12','2013-12-11
pst','2003-04-12 04:05:06');

-- View data.
openGauss=# SELECT * FROM time_type_tab;
   da    |     dai     |        dfgh         |          dfga          |         vbg
----------+-------------+---------------------+------------------------+---------------------
 21:21:21 | 21:21:21-08 | 2010-12-12 00:00:00 | 2013-12-11 16:00:00+08 | 2003-04-12 04:05:00
(1 row)

-- Delete the table.
openGauss=# DROP TABLE time_type_tab;

-- Create a table.
openGauss=# CREATE TABLE day_type_tab (a int,b INTERVAL DAY(3) TO SECOND (4));

-- Insert data.
openGauss=# INSERT INTO day_type_tab VALUES (1, INTERVAL '3' DAY);

-- View data.
openGauss=# SELECT * FROM day_type_tab;
 a |   b
---+--------
 1 | 3 days
(1 row)

-- Delete the table.
openGauss=# DROP TABLE day_type_tab;

-- Create a table.
openGauss=# CREATE TABLE year_type_tab(a int, b interval year (6));

-- Insert data.
openGauss=# INSERT INTO year_type_tab VALUES(1,interval '2' year);

-- View data.
openGauss=# SELECT * FROM year_type_tab;
 a |    b
---+---------
 1 | 2 years
(1 row)

-- Delete the table.
openGauss=# DROP TABLE year_type_tab;
```

Date Input

Date and time input is accepted in almost any reasonable formats, including ISO 8601, SQL-compatible, and traditional POSTGRES. The system allows you to customize the sequence of day, month, and year in the date input. Set the **DateStyle** parameter to **MDY** to select month-day-year interpretation, **DMY** to select day-month-year interpretation, or **YMD** to select year-month-day interpretation.

Remember that any date or time literal input needs to be enclosed with single quotation marks (''), and the syntax is as follows:

type [(p)] 'value'

The **p** that can be selected in the precision statement is an integer, indicating the number of fractional digits in the **seconds** column. Table 11-12 shows some possible inputs for the **date** type.

Table 11-12 Date input

Example	Description
1999-01-08	ISO 8601 (recommended format). January 8, 1999 in any mode
January 8, 1999	Unambiguous in any **datestyle** input mode
1/8/1999	January 8 in **MDY** mode. August 1 in **DMY** mode
1/18/1999	January 18 in **MDY** mode, rejected in other modes
01/02/03	● January 2, 2003 in **MDY** mode ● February 1, 2003 in **DMY** mode ● February 3, 2001 in **YMD** mode
1999-Jan-08	January 8 in any mode
Jan-08-1999	January 8 in any mode
08-Jan-1999	January 8 in any mode
99-Jan-08	January 8 in **YMD** mode, else error
08-Jan-99	January 8, except error in **YMD** mode
Jan-08-99	January 8, except error in **YMD** mode
19990108	ISO 8601. January 8, 1999 in any mode
990108	ISO 8601. January 8, 1999 in any mode
1999.008	Year and day of year
J2451187	Julian date
January 8, 99 BC	Year 99 BC

Example:

```
-- Create a table.
openGauss=# CREATE TABLE date_type_tab(coll date);

-- Insert data.
openGauss=# INSERT INTO date_type_tab VALUES (date '12-10-2010');

-- View data.
openGauss=# SELECT * FROM date_type_tab;
        coll
---------------------
 2010-12-10(1 row)

-- View the date format.
openGauss=# SHOW datestyle;
 DateStyle
-----------
 ISO, MDY
(1 row)

-- Set the date format.
openGauss=# SET datestyle='YMD';
SET

-- Insert data.
openGauss=# INSERT INTO date_type_tab VALUES(date '2010-12-11');

-- View data.
openGauss=# SELECT * FROM date_type_tab;
        coll
---------------------
 2010-12-10 2010-12-11(2 rows)

-- Delete the table.
openGauss=# DROP TABLE date_type_tab;
```

Time

The time-of-day types are **TIME [(p)] [WITHOUT TIME ZONE]** and **TIME [(p)] [WITH TIME ZONE]**. **TIME** alone is equivalent to **TIME WITHOUT TIME ZONE**.

If a time zone is specified in the input for **TIME WITHOUT TIME ZONE**, it is silently ignored.

For details about the time input types, see **Table 11-13**. For details about time zone input types, see **Table 11-14**.

Table 11-13 Time input

Example	Description
05:06.8	ISO 8601
4:05:06	ISO 8601
4:05	ISO 8601
40506	ISO 8601
4:05 AM	Same as 04:05. AM does not affect value

Example	Description
4:05 PM	Same as 16:05. Input hour must be <= 12
04:05:06.789-8	ISO 8601
04:05:06-08:00	ISO 8601
04:05-08:00	ISO 8601
040506-08	ISO 8601
04:05:06 PST	Time zone specified by abbreviation
2003-04-12 04:05:06 America/New_York	Time zone specified by full name

Table 11-14 Time zone input

Example	Description
PST	Abbreviation (for Pacific Standard Time)
America/New_York	Full time zone name
-8:00	ISO-8601 offset for PST
-800	ISO-8601 offset for PST
-8	ISO-8601 offset for PST

Example:

```
openGauss=# SELECT time '04:05:06';
   time
----------
 04:05:06
(1 row)

openGauss=# SELECT time '04:05:06 PST';
   time
----------
 04:05:06
(1 row)

openGauss=# SELECT time with time zone '04:05:06 PST';
   timetz
-------------
 04:05:06-08
(1 row)
```

Special Values

The special values supported by GaussDB are converted to common date/time values when being read. For details, see **Table 11-15**.

Table 11-15 Special values

Input String	Applicable Type	Description
epoch	date and timestamp	1970-01-01 00:00:00+00 (Unix system time zero)
infinity	timestamp	Later than any other timestamps
-infinity	timestamp	Earlier than any other timestamps
now	date, time, and timestamp	Start time of the current transaction
today	date and timestamp	Midnight today
tomorrow	date and timestamp	Midnight tomorrow
yesterday	date and timestamp	Midnight yesterday
allballs	time	00:00:00.00 UTC

Interval Input

The input of **reltime** can be any valid interval in text format. It can be a number (negative numbers and decimals are also allowed) or a specific time, which must be in SQL standard format, ISO-8601 format, or POSTGRES format. In addition, the text input needs to be enclosed with single quotation marks ('').

For details, see **Table 6 Interval input**.

Table 11-16 Interval input

Input	Output	Description
60	2 mons	Numbers are used to indicate intervals. The default unit is day. Decimals and negative numbers are allowed. Particularly, a negative interval syntactically means how long before.
31.25	1 mons 1 days 06:00:00	
-365	-12 mons -5 days	
1 years 1 mons 8 days 12:00:00	1 years 1 mons 8 days 12:00:00	Intervals are in POSTGRES format. They can contain both positive and negative numbers and are case-insensitive. Output is a simplified POSTGRES interval converted from the input.
-13 months -10 hours	-1 years -25 days -04:00:00	
-2 YEARS +5 MONTHS 10 DAYS	-1 years -6 mons -25 days -06:00:00	

Input	Output	Description
P-1.1Y10M	-3 mons -5 days -06:00:00	Intervals are in ISO-8601 format. They can contain both positive and negative numbers and are case-insensitive. Output is a simplified POSTGRES interval converted from the input.
-12H	-12:00:00	

Example:

```
-- Create a table.
openGauss=# CREATE TABLE reltime_type_tab(col1 character(30), col2 reltime);

-- Insert data.
openGauss=# INSERT INTO reltime_type_tab VALUES ('90', '90');
openGauss=# INSERT INTO reltime_type_tab VALUES ('-366', '-366');
openGauss=# INSERT INTO reltime_type_tab VALUES ('1975.25', '1975.25');
openGauss=# INSERT INTO reltime_type_tab VALUES ('-2 YEARS +5 MONTHS 10 DAYS', '-2 YEARS +5
MONTHS 10 DAYS');
openGauss=# INSERT INTO reltime_type_tab VALUES ('30 DAYS 12:00:00', '30 DAYS 12:00:00');
openGauss=# INSERT INTO reltime_type_tab VALUES ('P-1.1Y10M', 'P-1.1Y10M');

-- View data.
openGauss=# SELECT * FROM reltime_type_tab;
             col1              |                col2
--------------------------------+-------------------------------------
 1975.25                        | 5 years 4 mons 29 days
 -2 YEARS +5 MONTHS 10 DAYS     | -1 years -6 mons -25 days -06:00:00
 P-1.1Y10M                      | -3 mons -5 days -06:00:00
 -366                           | -1 years -18:00:00
 90                             | 3 mons
 30 DAYS 12:00:00               | 1 mon 12:00:00
(6 rows)

-- Delete the table.
openGauss=# DROP TABLE reltime_type_tab;
```

11.3.7 Geometric

Table 11-17 lists the geometric types that can be used in GaussDB. The most fundamental type, the point, forms the basis for all of the other types.

Table 11-17 Geometric types

Name	Storage Space	Description	Representation
point	16 bytes	Point on a plane	(x,y)
lseg	32 bytes	Finite line segment	((x1,y1),(x2,y2))
box	32 bytes	Rectangle	((x1,y1),(x2,y2))
path	16+16n bytes	Closed path (similar to polygon)	((x1,y1),...)

Name	Storage Space	Description	Representation
path	16+16n bytes	Open path	[(x1,y1),...]
polygon	40+16n bytes	Polygon (similar to closed path)	((x1,y1),...)
circle	24 bytes	Circle	<(x,y),r> (center point and radius)

A rich set of functions and operators is available in GaussDB to perform various geometric operations, such as scaling, translation, rotation, and determining intersections. For details, see **Geometric Functions and Operators**.

Points

Points are the fundamental two-dimensional building block for geometric types. Values of the **point** type are specified using either of the following syntax:

```
( x , y )
x , y
```

where x and y are the respective coordinates, as floating-point numbers.

Points are output using the first syntax.

Line Segments

Line segments (**lseg**) are represented by pairs of points. Values of the **lseg** type are specified using any of the following syntax:

```
[ ( x1 , y1 ) , ( x2 , y2 ) ]
( ( x1 , y1 ) , ( x2 , y2 ) )
( x1 , y1 ) , ( x2 , y2 )
x1 , y1   ,   x2 , y2
```

where (x1,y1) and (x2,y2) are the end points of the line segment.

Line segments are output using the first syntax.

Boxes

Boxes are represented by pairs of points that are opposite corners of the box. Values of the **box** type are specified using any of the following syntax:

```
( ( x1 , y1 ) , ( x2 , y2 ) )
( x1 , y1 ) , ( x2 , y2 )
x1 , y1   ,   x2 , y2
```

where (x1,y1) and (x2,y2) are any two opposite corners of the box.

Boxes are output using the second syntax.

Any two opposite corners can be supplied on input, but in this order, the values will be reordered as needed to store the upper right and lower left corners.

Paths

Paths are represented by lists of connected points. Paths can be open, where the first and last points in the list are considered not connected, or closed, where the first and last points are considered connected.

Values of the **path** type are specified using any of the following syntax:

```
[ ( x1 , y1 ) , ... , ( xn , yn ) ]
( ( x1 , y1 ) , ... , ( xn , yn ) )
( x1 , y1 ) , ... , ( xn , yn )
( x1 , y1   , ... ,   xn , yn )
x1 , y1   , ... ,   xn , yn
```

where the points are the end points of the line segments comprising the path. Square brackets ([]) indicate an open path, while parentheses (()) indicate a closed path. When the outermost parentheses are omitted, as in the third through fifth syntax, a closed path is assumed.

Paths are output using the first or second syntax.

Polygons

Polygons are represented by lists of points (the vertexes of the polygon). Polygons are very similar to closed paths, but are stored differently and have their own set of support functions.

Values of the **polygon** type are specified using any of the following syntax:

```
( ( x1 , y1 ) , ... , ( xn , yn ) )
( x1 , y1 ) , ... , ( xn , yn )
( x1 , y1   , ... ,   xn , yn )
x1 , y1   , ... ,   xn , yn
```

where the points are the end points of the line segments comprising the boundary of the polygon.

Polygons are output using the first syntax.

Circles

Circles are represented by a center point and radius. Values of the **circle** type are specified using any of the following syntax:

```
< ( x , y ) , r >
( ( x , y ) , r )
( x , y ) , r
x , y   , r
```

where **(x,y)** is the center point and **r** is the radius of the circle.

Circles are output using the first syntax.

11.3.8 Network Address Types

GaussDB offers data types to store IPv4 and MAC addresses.

It is better to use these types instead of plain text types to store network addresses, because these types offer input error checking and specialized operators and functions (see **Network Address Functions and Operators**).

Table 11-18 Network address types

Name	Storage Space	Description
cidr	7 bytes	IPv4 networks
inet	7 bytes	IPv4 hosts and networks
macaddr	6 bytes	MAC address

cidr

The **cidr** type (Classless Inter-Domain Routing) holds an IPv4 network address. The format for specifying networks is **address/y** where **address** is the network represented as an IPv4 address, and **y** is the number of bits in the netmask. If **y** is omitted, it is calculated using assumptions from the older classful network numbering system, except it will be at least large enough to include all of the octets written in the input.

Table 11-19 cidr type input examples

cidr Input	cidr Output	abbrev(cidr)
192.168.100.128/25	192.168.100.128/25	192.168.100.128/25
192.168/24	192.168.0.0/24	192.168.0/24
192.168/25	192.168.0.0/25	192.168.0.0/25
192.168.1	192.168.1.0/24	192.168.1/24
192.168	192.168.0.0/24	192.168.0/24
10.1.2	10.1.2.0/24	10.1.2/24
10.1	10.1.0.0/16	10.1/16
10	10.0.0.0/8	10/8
10.1.2.3/32	10.1.2.3/32	10.1.2.3/32

inet

The **inet** type holds an IPv4 host address, and optionally its subnet, all in one field. The subnet is represented by the number of network address bits present in the host address (the "netmask"). If the netmask is 32 and the address is an IPv4 address, then the value does not indicate a subnet, only a single host.

The input format for this type is **address/y** where **address** is an IPv4 address and **y** is the number of bits in the netmask. If **/y** is omitted, the subnet mask is **32** for an IPv4 address, and the value represents just a single host. On display, the **/y** portion is suppressed if the netmask specifies a single host.

The essential difference between the **inet** and **cidr** data types is that **inet** accepts values with nonzero bits to the right of the netmask, whereas **cidr** does not.

macaddr

The **macaddr** type stores MAC addresses, known for example from Ethernet card hardware addresses (although MAC addresses are used for other purposes as well). Input is accepted in the following formats:

```
'08:00:2b:01:02:03'
'08-00-2b-01-02-03'
'08002b:010203'
'08002b-010203'
'0800.2b01.0203'
'08002b010203'
```

These examples would all specify the same address. Upper and lower cases are accepted for the digits a through f. Output is always in the first of the forms shown.

11.3.9 Bit String Types

Bit strings are strings of 1's and 0's. They can be used to store bit masks.

GaussDB supports two bit string types: bit(n) and bit varying(n), in which **n** is a positive integer.

The **bit** type data must match the length *n* exactly. It is an error to attempt to store shorter or longer bit strings. The **bit varying** data is of variable length up to the maximum length *n*; longer strings will be rejected. Writing **bit** without a length is equivalent to **bit(1)**, while **bit varying** without a length specification means unlimited length.

NOTE

If one explicitly casts a bit-string value to **bit(n)**, it will be truncated or zero-padded on the right to be exactly *n* bits, without raising an error.

Similarly, if one explicitly casts a bit-string value to **bit varying(n)**, it will be truncated on the right if it is more than *n* bits.

```
-- Create a table.
openGauss=# CREATE TABLE bit_type_t1
(
    BT_COL1 INTEGER,
    BT_COL2 BIT(3),
    BT_COL3 BIT VARYING(5)
) ;

-- Insert data.
openGauss=# INSERT INTO bit_type_t1 VALUES(1, B'101', B'00');

-- Specify the type length. An error is reported if an inserted string exceeds this length.
openGauss=# INSERT INTO bit_type_t1 VALUES(2, B'10', B'101');
ERROR:  bit string length 2 does not match type bit(3)
CONTEXT:  referenced column: bt_col2

-- Specify the type length. Data is converted if it exceeds this length.
openGauss=# INSERT INTO bit_type_t1 VALUES(2, B'10'::bit(3), B'101');

-- View data.
openGauss=# SELECT * FROM bit_type_t1;
 bt_col1 | bt_col2 | bt_col3
---------+---------+---------
```

```
      1 | 101     | 00
      2 | 100     | 101
(2 rows)

-- Delete the table.
openGauss=# DROP TABLE bit_type_t1;
```

11.3.10 Text Search Types

GaussDB offers two data types that are designed to support full text search. The **tsvector** type represents a document in a form optimized for text search. The **tsquery** type similarly represents a text query.

tsvector

The **tsvector** type represents a retrieval unit, usually a textual column within a row of a database table, or a combination of such columns. A **tsvector** value is a sorted list of distinct lexemes, which are words that have been normalized to merge different variants of the same word. Sorting and deduplication are done automatically during input. The **to_tsvector** function is used to parse and normalize a document string.

A **tsvector** value is a sorted list of distinct lexemes, which are words that have been formatted different entries. During segmentation, **tsvector** automatically performs duplicate-elimination to the entries for input in a certain order. Example:

```
openGauss=# SELECT 'a fat cat sat on a mat and ate a fat rat'::tsvector;
                    tsvector
----------------------------------------------------
 'a' 'and' 'ate' 'cat' 'fat' 'mat' 'on' 'rat' 'sat'
(1 row)
```

It can be seen from the preceding example that **tsvector** segments a string by spaces, and segmented lexemes are sorted based on their length and alphabetical order. To represent lexemes containing whitespace or punctuation, surround them with quotation marks:

```
openGauss=# SELECT $$the lexeme '    ' contains spaces$$::tsvector;
                 tsvector
-------------------------------------------
 '    ' 'contains' 'lexeme' 'spaces' 'the'
(1 row)
```

Use double dollar signs ($$) to mark entries containing single quotation marks ('').

```
openGauss=# SELECT $$the lexeme 'Joe''s' contains a quote$$::tsvector;
                    tsvector
------------------------------------------------
 'Joe''s' 'a' 'contains' 'lexeme' 'quote' 'the'
(1 row)
```

Optionally, integer positions can be attached to lexemes:

```
openGauss=# SELECT 'a:1 fat:2 cat:3 sat:4 on:5 a:6 mat:7 and:8 ate:9 a:10 fat:11 rat:12'::tsvector;
                                  tsvector
-------------------------------------------------------------------------------
 'a':1,6,10 'and':8 'ate':9 'cat':3 'fat':2,11 'mat':7 'on':5 'rat':12 'sat':4
(1 row)
```

A position normally indicates the source word's location in the document. Positional information can be used for proximity ranking. Position values range from 1 to 16383. The maximum value is **16383**. Duplicate positions for the same lexeme are discarded.

Lexemes that have positions can further be labeled with a weight, which can be **A**, **B**, **C**, or **D**. **D** is the default and therefore is not shown in output.

```
openGauss=# SELECT 'a:1A fat:2B,4C cat:5D'::tsvector;
          tsvector
----------------------------
 'a':1A 'cat':5 'fat':2B,4C
(1 row)
```

Weights are typically used to reflect document structure, for example, by marking title words differently from body words. Text search ranking functions can assign different priorities to the different weight markers.

The following example is the standard usage of the **tsvector** type. Example:

```
openGauss=# SELECT 'The Fat Rats'::tsvector;
      tsvector
--------------------
 'Fat' 'Rats' 'The'
(1 row)
```

For most English-text-searching applications, the above words would be considered non-normalized, which should usually be passed through **to_tsvector** to normalize the words appropriately for searching:

```
openGauss=# SELECT to_tsvector('english', 'The Fat Rats');
   to_tsvector
-----------------
 'fat':2 'rat':3
(1 row)
```

tsquery

The **tsquery** type represents a retrieval condition. A **tsquery** value stores lexemes that are to be searched for, and combines them honoring the **Boolean** operators **& (AND)**, **| (OR)**, and **! (NOT)**. Parentheses can be used to enforce grouping of the operators. The **to_tsquery** and **plainto_tsquery** functions will normalize lexemes before the lexemes are converted to the **tsquery** type.

```
openGauss=# SELECT 'fat & rat'::tsquery;
    tsquery
---------------
 'fat' & 'rat'
(1 row)

openGauss=# SELECT 'fat & (rat | cat)'::tsquery;
          tsquery
---------------------------
 'fat' & ( 'rat' | 'cat' )
(1 row)

openGauss=# SELECT 'fat & rat & ! cat'::tsquery;
        tsquery
------------------------
 'fat' & 'rat' & !'cat'
(1 row)
```

In the absence of parentheses, **! (NOT)** binds most tightly, and **& (AND)** binds more tightly than **| (OR)**.

Lexemes in a **tsquery** can be labeled with one or more weight letters, which restrict them to match only **tsvector** lexemes with matching weights:

```
openGauss=# SELECT 'fat:ab & cat'::tsquery;
    tsquery
------------------
 'fat':AB & 'cat'
(1 row)
```

Also, lexemes in a **tsquery** can be labeled with * to specify prefix matching:

```
openGauss=# SELECT 'super:*'::tsquery;
  tsquery
-----------
 'super':*
(1 row)
```

This query will match any word in a **tsvector** that begins with "super".

Note that prefixes are first processed by text search configurations, which means that the following example returns true:

```
openGauss=# SELECT to_tsvector( 'postgraduate' ) @@ to_tsquery( 'postgres:*' ) AS RESULT;
  result
----------
 t
(1 row)
```

This is because **postgres** gets stemmed to **postgr**:

```
openGauss=# SELECT to_tsquery('postgres:*');
 to_tsquery
------------
 'postgr':*
(1 row)
```

It then matches **postgraduate**.

'Fat:ab & Cats' is normalized to the **tsquery** type as follows:

```
openGauss=# SELECT to_tsquery('Fat:ab & Cats');
   to_tsquery
------------------
 'fat':AB & 'cat'
(1 row)
```

11.3.11 UUID

The data type **UUID** stores Universally Unique Identifiers (UUID) as defined by RFC 4122, ISO/IEF 9834-8:2005, and related standards. This identifier is a 128-bit quantity that is generated by an algorithm chosen to make it very unlikely that the same identifier will be generated by anyone else in the known universe using the same algorithm.

A UUID is written as a sequence of lower-case hexadecimal digits, in several groups separated by hyphens, specifically a group of 8 digits followed by three groups of 4 digits followed by a group of 12 digits, for a total of 32 digits representing the 128 bits. An example of a UUID in this standard form is:

```
a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11
```

GaussDB also accepts the following alternative forms for input: use of upper-case letters and digits, the standard format surrounded by braces, omitting some or all hyphens, adding a hyphen after any group of four digits. An example is provided as follows:

```
A0EEBC99-9C0B-4EF8-BB6D-6BB9BD380A11
{a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11}
```

```
a0eebc999c0b4ef8bb6d6bb9bd380a11
a0ee-bc99-9c0b-4ef8-bb6d-6bb9-bd38-0a11
```

Output is always in the standard form.

11.3.12 JSON/JSONB Types

JavaScript Object Notation (JSON) data can be a single scalar, an array, or a key-value pair object. The array and object can be called a container:

- Scalar: a number, Boolean, string, or null
- Array: defined in a pair of square brackets ([]), in which elements can be any type of JSON data, and are not necessarily of the same type.
- Object: defined in a pair of braces ({}), in which objects are stored in the format of **key:value**. Each key must be a string enclosed by a pair of double quotation marks (""), and its value can be any type of JSON data. In case of duplicate keys, the last key-value pair will be used.

GaussDB supports two types JSON and JSONB to store JSON data. JSON data is a complete copy of the input, retaining the entered spaces, duplicate keys, and sequence, while JSONB stores data in a decomposed binary form, removing semantic-irrelevant details and duplicate keys, and sorting key-values. Therefore, JSONB data does not need to be parsed.

Basically, both are JSON data types. However, they differ greatly in efficiency. Because JSON data is an exact copy of the input text, the data must be parsed on every execution; in contrast, JSONB data is stored in a decomposed binary form and can be processed faster, though this makes it slightly slower to input due to the conversion mechanism. In addition, because the JSONB data form is unified, it better supports more powerful functions, for example, comparing sizes according to a specific rule. JSONB also supports indexing, which is a significant advantage.

Input Format

An input must be a JSON-compliant string, which is enclosed in single quotation marks ('').

Null (null-json): Only null is supported, and all letters are in lowercase.

```
select 'null'::json;   -- suc
select 'NULL'::jsonb;  -- err
```

Number (num-json): The value can be a positive or negative integer, decimal fraction, or 0. The scientific notation is supported.

```
select '1'::json;select '-1.5'::json;select '-1.5e-5'::jsonb, '-1.5e+2'::jsonb;select '001'::json, '+15'::json, 'NaN'::json;
-- Redundant leading zeros, plus signs (+), NaN, and infinity are not supported.
```

Boolean (bool-json): The value can only be **true** or **false** in lowercase.

```
select 'true'::json;select 'false'::jsonb;
```

String (str-json): The value must be a string enclosed in double quotation marks ("").

```
select '"a"'::json;select '"abc"'::jsonb;
```

Array (array-json): Arrays are enclosed in square brackets ([]). Elements in the array can be any valid JSON data, and are unnecessarily of the same type.

```
select '[1, 2, "foo", null]'::json;select '[]'::json;select '[1, 2, "foo", null, [[]], {}]'::jsonb;
```

Object (object-json): The value is enclosed in braces ({}). The key must be a JSON-compliant string, and the value can be any valid JSON string.

```
select '{}'::json;select '{"a": 1, "b": {"a": 2,  "b": null}}'::json;select '{"foo": [true, "bar"], "tags": {"a": 1, "b":
null}}'::jsonb;
```

CAUTION

- Note that 'null'::json and null::json are different, which are similar to the strings str="" and str=null.
- For numbers, when scientific notation is used, JSONB expands them, while JSON stores an exact copy of the input text.

JSONB Advanced Features

- Precautions
  - Row-store tables are not supported.
  - It cannot be used as a partition key.
  - Foreign tables and MOTs are not supported.

The main difference between JSON and JSONB lies in the storage mode. JSONB stores parsed binary data, which reflects the JSON hierarchy and facilitates direct access. Therefore, JSONB has many advanced features that JSON does not have.

- Format normalization
  - After the input object-json string is parsed into JSONB binary, semantically irrelevant details are naturally discarded, for example, spaces:
    ```
    openGauss=# select '   [1, " a ", {"a"   :1    }]  '::jsonb;
           jsonb
    ----------------------
     [1, " a ", {"a": 1}]
    (1 row)
    ```
  - For object-json, duplicate key-values are deleted and only the last key-value is retained. For example:
    ```
    openGauss=# select '{"a" : 1, "a" : 2}'::jsonb;
      jsonb
    ----------
     {"a": 2}
    (1 row)
    ```
  - For object-json, key-values will be re-sorted. The sorting rule is as follows: 1. Longer key-values are sorted last. 2. If the key-values are of the same length, the key-values with a larger ASCII code are sorted after the key-values with a smaller ASCII code:
    ```
    openGauss=# select '{"aa" : 1, "b" : 2, "a" : 3}'::jsonb;
              jsonb
    ---------------------------
    {"a": 3, "b": 2, "aa": 1}
    (1 row)
    ```
- Size comparison

  Format normalization ensures that only one form of JSONB data exists in the same semantics. Therefore, sizes may be compared according to a specific rule.

- First, type comparison: **object-jsonb** > **array-jsonb** > **bool-jsonb** > **num-jsonb** > **str-jsonb** > **null-jsonb**
- Content comparison if the data type is the same:
  - **str-json**: The default text sorting rule of the database is used for comparison. A positive value indicates greater than, a negative value indicates less than, and **0** indicates equal.
  - **num-json**: numeric comparison
  - **bool-json**: **true** > **false**
  - **array-jsonb**: long elements > short elements. If the lengths are the same, compare each element in sequence.
  - **object-jsonb**: If the length of a key-value pair is longer than that of a short key-value pair, the key is compared first, and then the value is compared.

CAUTION

For comparison within the **object-jsonb** type, the final result after format sorting is used for comparison. Therefore, the comparison result may not be intuitive compared with the direct input.

- Creating indexes, primary keys, and foreign keys
  - B-tree index

    B-tree indexes, primary keys, and foreign keys can be created for the **JSONB** type.
  - GIN index

    GIN indexes can be used to effectively search for keys or key-value pairs that appear in a large number of JSONB documents (datums). Two GIN operator classes (**jsonb_ops** and **jsonb_hash_ops**) are provided for different performance and flexibility choices. The default GIN operator class supports **@>**, **<@**, **?**, **?&** and **?|** operator query. The non-default GIN operator class **jsonb_path_ops** supports only the **@>** and **<@** operators.

    For details about the operators, see **JSON/JSONB Functions and Operators**.
- Inclusion and existence

  Querying whether a JSON contains some elements or whether some elements exist in a JSON is an important capability of JSONB.

```
-- A simple scalar/original value contains only the same value: SELECT '"foo"'::jsonb @> '"foo"'::jsonb; -- The array on the left contains the string on the right. SELECT' [ 1, "aa", 3 ] '::jsonb? 'aa'; -- The array on the left contains all elements in the array on the right. The sequence and repetition are not specified. SELECT '[1, 2, 3]'::jsonb @> '[1, 3, 1]'::jsonb; -- The object-json on the left contains all key-values of the object-json on the right. SELECT '{"product": "PostgreSQL", "version": 9.4, "jsonb":true}'::jsonb @> '{"version":9.4}'::jsonb; -- The array on the left does not contain all elements of the array on the right, because the three elements of the array on the left are 1, 2, and [1,3], and the elements on the right are 1 and 3. SELECT'[1, 2, [1, 3] ]. '::jsonb @>' [1, 3] '::jsonb; -- Produces a false. SELECT '{"foo": {"bar": "baz"}}'::jsonb @> '{"bar": "baz"}'::jsonb; -- Similarly, false is produced.
```

  For details about the operators, see **JSON/JSONB Functions and Operators**.

- Functions and operators

  For details about the functions and operators supported by the JSON/JSONB type, see **JSON/JSONB Functions and Operators**.

11.3.13 HLL

HyperLoglog (HLL) is an approximation algorithm for efficiently counting the number of distinct values in a data set. It features faster computing and lower space usage. You only need to store HLL data structures, instead of data sets. When new data is added to a data set, make hash calculation on the data and insert the result to an HLL. Then, you can obtain the final result based on the HLL.

Table 11-20 compares HLL with other algorithms.

Table 11-20 Comparison between HLL and other algorithms

Item	Sorting Algorithm	Hash Algorithm	HLL
Time complexity	O(nlogn)	O(n)	O(n)
Space complexity	O(n)	O(n)	log(logn)
Error rate	0	0	≈0.8%
Storage space requirement	Size of original data	Size of original data	The maximum size is 16 KB by default.

HLL has advantages over others in the computing speed and storage space requirement. In terms of time complexity, the sorting algorithm needs O(nlogn) time for sorting, and the hash algorithm and HLL need O(n) time for full table scanning. In terms of storage space requirements, the sorting algorithm and hash algorithm need to store raw data before collecting statistics, whereas the HLL algorithm needs to store only the HLL data structures rather than the raw data, and thereby occupying a fixed space of about 16 KB.

NOTICE

- In the current default specifications, the maximum number of distinct values that can be calculated is about 1.1e + 15, and the error rate is 0.8%. If the calculation result exceeds the maximum, the error rate of the calculation result will increase, or the calculation will fail and an error will be reported.
- When using this feature for the first time, you need to evaluate the distinct values of the service, properly select configuration parameters, and perform verification to ensure that the accuracy meets requirements.
  - By default, the distinct value is 1.1e + 15. If the distinct value is NaN, you need to adjust log2m or use another algorithm to calculate the distinct value.
  - The hash algorithm has an extremely low probability of collision. However, you are still advised to select 2 or 3 hash seeds for verification when using the hash algorithm for the first time. If there is only a small difference between the distinct values, you can select any one of the seeds as the hash seed.

Table 11-21 describes main HLL data structures.

Table 11-21 Main HLL data structures

Data Type	Function
hll	The HLL header is a 27-byte field. By default, the data length ranges from 0 KB to 16 KB. The distinct value can be obtained.

When you create an HLL data type, 0 to 4 input parameters are supported. The parameter meanings and specifications are the same as those of the **hll_empty** function. The first parameter is **log2m**, indicating the logarithm of the number of buckets, and its value ranges from 10 to 16. The second parameter is **log2explicit**, indicating the threshold in explicit mode, and its value ranges from 0 to 12. The third parameter is **log2sparse**, indicating the threshold of the Sparse mode, and its value ranges from 0 to 14. The fourth parameter is **duplicatecheck**, indicating whether to enable duplicatecheck, and its value ranges from 0 to 1. When the input parameter is set to **–1**, the default value of the HLL parameter is used. You can run the **\d** or **\d+** command to view the parameters of the HLL type.

NOTE

When the HLL data type is created, the result varies depending on the input parameter behavior:

- When creating an HLL type, do not set the input parameter or set it to **–1**. Use the default value of the corresponding HLL parameter.
- If a valid value is set for the input parameter, the corresponding HLL parameter uses the input value.
- If the input value is invalid, an error is reported when the HLL type is created.

```
-- Create an HLL table without specifying input parameters.
openGauss=# create table t1 (id integer, set hll);
```

```
openGauss=# \d t1
      Table "public.t1"
 Column |  Type   | Modifiers
--------+---------+-----------
 id     | integer |
 set    | hll     |

-- Create an HLL table, specify the first two input parameters, and use the default values for the last two
input parameters.
openGauss=# create table t2 (id integer, set hll(12,4));
openGauss=# \d t2
          Table "public.t2"
 Column |      Type      | Modifiers
--------+----------------+-----------
 id     | integer        |
 set    | hll(12,4,12,0) |

-- Create an HLL table, specify the third input parameter, and use default values for other parameters.
openGauss=# create table t3(id int, set hll(-1,-1,8,-1));
openGauss=# \d t3
          Table "public.t3"
 Column |      Type      | Modifiers
--------+----------------+-----------
 id     | integer        |
 set    | hll(14,10,8,0) |

-- When a user creates an HLL table and specifies an invalid input parameter, an error is reported.
openGauss=# create table t4(id int, set hll(5,-1));
ERROR:  log2m = 5 is out of range, it should be in range 10 to 16, or set -1 as default
```

NOTE

When inserting an HLL object to an HLL table, ensure that the parameters of the HLL type are the same as those of the inserted object. Otherwise, an error is reported.

```
-- Create an HLL table:
openGauss=# create table t1(id integer, set hll(14));

-- Insert an HLL object to a table. The insertion succeeds because parameter types are consistent.
openGauss=# insert into t1 values (1, hll_empty(14,-1));

-- Insert an HLL object to a table. The insertion fails because parameter types are inconsistent.
openGauss=# insert into t1(id, set) values (1, hll_empty(14,5));
ERROR:  log2explicit does not match: source is 5 and dest is 10
```

The following describes HLL application scenarios.

- Scenario 1: "Hello World"

  The following example shows how to use the HLL data type:

  ```
  -- Create a table with the HLL type:
  openGauss=# create table helloworld (id integer, set hll);

  -- Insert an empty HLL to the table:
  openGauss=# insert into helloworld(id, set) values (1, hll_empty());

  -- Add a hashed integer to the HLL:
  openGauss=# update helloworld set set = hll_add(set, hll_hash_integer(12345)) where id = 1;

  -- Add a hashed string to the HLL:
  openGauss=# update helloworld set set = hll_add(set, hll_hash_text('hello world')) where id = 1;

  -- Obtain the number of distinct values of the HLL:
  openGauss=# select hll_cardinality(set) from helloworld where id = 1;
   hll_cardinality
  -----------------
                 2
  (1 row)
  ```

```
-- Delete the table.
openGauss=#  drop table helloworld;
```

- Scenario 2: Collect statistics about website visitors.

  The following example shows how an HLL collects statistics on the number of users visiting a website within a period of time:

```
-- Create a raw data table to show that a user has visited the website at a certain time:
openGauss=# create table facts (
         date            date,
         user_id         integer
);

 -- Create a raw data table to show that a user has visited the website at a certain time:
openGauss=# insert into facts values ('2019-02-20', generate_series(1,100));
openGauss=# insert into facts values ('2019-02-21', generate_series(1,200));
openGauss=# insert into facts values ('2019-02-22', generate_series(1,300));
openGauss=# insert into facts values ('2019-02-23', generate_series(1,400));
openGauss=# insert into facts values ('2019-02-24', generate_series(1,500));
openGauss=# insert into facts values ('2019-02-25', generate_series(1,600));
openGauss=# insert into facts values ('2019-02-26', generate_series(1,700));
openGauss=# insert into facts values ('2019-02-27', generate_series(1,800));

-- Create another table and specify an HLL column:
openGauss=# create table daily_uniques (
    date            date UNIQUE,
    users           hll
);

-- Group data by date and insert the data into the HLL:
openGauss=# insert into daily_uniques(date, users)
    select date, hll_add_agg(hll_hash_integer(user_id))
    from facts
    group by 1;

-- Calculate the numbers of users visiting the website every day:
openGauss=# select date, hll_cardinality(users) from daily_uniques order by date;
    date    | hll_cardinality
------------+------------------
 2019-02-20 |              100
 2019-02-21 | 200.217913059312
 2019-02-22 |  301.76494508014
 2019-02-23 | 400.862858326446
 2019-02-24 | 502.626933349694
 2019-02-25 | 601.922606454213
 2019-02-26 | 696.602316769498
 2019-02-27 | 798.111731634412
(8 rows)

-- Calculate the number of users who had visited the website in the week from February 20, 2019 to
February 26, 2019:
openGauss=# select hll_cardinality(hll_union_agg(users)) from daily_uniques where date >=
'2019-02-20'::date and date <= '2019-02-26'::date;
 hll_cardinality
------------------
 702.941844662509
(1 row)

-- Calculate the number of users who had visited the website yesterday but have not visited the
website today:
openGauss=# SELECT date, (#hll_union_agg(users) OVER two_days) - #users AS lost_uniques FROM
daily_uniques WINDOW two_days AS (ORDER BY date ASC ROWS 1
PRECEDING);
    date    | lost_uniques
------------+--------------
 2019-02-20 |            0
 2019-02-21 |            0
 2019-02-22 |            0
 2019-02-23 |            0
 2019-02-24 |            0
 2019-02-25 |           0
 2019-02-26 |           0
 2019-02-27 |           0
(8 rows)

-- Delete the table.
openGauss=# drop table facts;
openGauss=# drop table daily_uniques;
```

- Scenario 3: The data to be inserted does not meet the requirements of the HLL data structure.

  When inserting data into a column of the HLL type, ensure that the data meets the requirements of the HLL data structure. If the data does not meet the requirements after being parsed, an error will be reported. In the following example, **E\\1234** to be inserted does not meet the requirements of the HLL data structure after being parsed. As a result, an error is reported.

```
openGauss=# create table test(id integer, set hll);
openGauss=# insert into test values(1, 'E\\1234');
ERROR:  not a hll type, size=6 is not enough
openGauss=# drop table test;
```

11.3.14 Range

A range type is a data type that represents the range of a value of an element type (called the subtype of a range). For example, the range of timestamp may be used to express a time range in which a conference room is reserved. In this case, the data type is tsrange and timestamp is the subtype. The subtype must have an overall order so that the element value can be clearly specified within a range, before, or after.

Range types are useful because they can express multiple element values in a single range value and can clearly express concepts such as range overlapping. The time and date range used for scheduling is the best example, as the range of an instrument are also examples of range type.

Built-in Range

The following built-in ranges are available:

- int4range: integer range.
- int8range: bigint range.
- numrange: numeric range.
- tsrange: range of timestamp without the time zone.
- tstzrange: range of timestamp with the time zone
- daterange: date range.

In addition, you can define your own range types. For details, see **CREATE TYPE**.

Example

```
CREATE TABLE reservation (room int, during tsrange);
INSERT INTO reservation VALUES (1108, '[2010-01-01 14:30, 2010-01-01 15:30)');
-- Inclusion
SELECT int4range(10, 20) @> 3;
-- Overlapping
SELECT numrange(11.1, 22.2) && numrange(20.0, 30.0);
-- Upper bound extraction
SELECT upper(int8range(15, 25));
```

```
-- Intersection set
SELECT int4range(10, 20) * int4range(15, 25);
-- Empty range
SELECT isempty(numrange(1, 5));
```

See the complete list of operators and functions on a range type in **Range Functions and Operators**.

Including and Excluding Bounds

Each non-empty range has two bounds, a lower bound and an upper bound. All values between the upper and lower bounds are included in the range. An inclusion bound means that the bound value itself is included in the range, while an exclusion bound means that the bound value is not included in the range.

In a textual form of a range, the inclusion lower bound is expressed as "[" and an exclusion lower bound is expressed as "(". Similarly, one containing the upper bound is expressed as "]" and one excluding the upper bound is expressed as ")" (for details, see **Range Input/Output**).

The lower_inc and upper_inc functions test the upper and lower bounds of a range value, respectively.

Infinite (Unbounded) Range

When the lower bound of a range is unbounded, it means that all values less than the upper bound are included in the range, for example, (,3] meaning all values less than the upper bound 3 are included in the range. Similarly, when the upper bound of a range is unbounded, all values greater than the upper bound are included in the range. When both the upper and lower bounds are unbounded, all values of the element type are considered within the range. The missing bounds are automatically converted to exclusions, for example, [,] is converted to (,). You can think of these missing values as positive infinity or negative infinity, but they are special range type values and are considered to be positive and negative infinity values that go beyond any range element type.

Element types with the infinity values can be used as explicit bound values. For example, in the timestamp range, [today, infinity) does not include a special timestamp value infinity.

The lower_inf and upper_inf functions test the infinite upper and lower bounds of a range, respectively.

Range Input/Output

The input of a range value must follow one of the following formats:

```
(lower-bound, upper-bound)
(lower-bound, upper-bound]
[lower-bound, upper-bound)
[lower-bound, upper-bound]
Empty
```

Parentheses () or square brackets [] indicate whether the upper and lower bounds are excluded or included. Note that the last format is empty, which represents an empty range (a range that does not contain values).

The value of *lower-bound* can be a valid input character string of the subtype or null, indicating that there is no lower bound. Similarly, *upper-bound* can be a valid input character string of the subtype or null, indicating that there is no upper bound.

Each bound value can be referenced using the quotation marks("") character. This is necessary if the bounds value contains parentheses (), square brackets [], commas (,), quotation marks (""), or backslashes (\), because otherwise those characters will be considered part of the range syntax. To put the quotation mark or backslash in a referenced bound value, put a backslash in front of it (and a pair of double quotation marks in its quoted bound value represents one quotation mark character, which is similar to the single quotation mark rule in SQL character strings). In addition, you can avoid referencing and use backslash escapes to protect all data characters, otherwise they will be used as part of the return syntax. Also, if you want to write a bound value that is an empty string, write **""**, indicating infinite bounds.

Spaces are allowed before and after a range value, but any space between parentheses() or square brackets[] is used as part of the upper or lower bound value (depending on the element type, the space may or may not represent a value).

Examples:

```
-- 3 is included, 7 is not included, and all values between 3 and 7 are included.
SELECT '[3,7)'::int4range;
-- Neither 3 nor 7 is included, but all values between them are included.
SELECT '(3,7)'::int4range;
-- Only value 4 is included.
SELECT '[4,4]'::int4range;
-- Exclude any value (and will be normalized to empty)
SELECT '[4,4)'::int4range;
```

Constructing Range

Each range type has a constructor function with the same name. Using constructor functions is often more convenient than writing a range literal constant because it avoids extra references to bound values. Constructor functions accept two or three parameters. Two parameters form a range in the standard form, where the lower bound is included and the upper bound is excluded, and three parameters form a range according to the bound specified by the third parameter. The third parameter must be one of the following character strings: (), (], [], or []. For example:

```
-- The complete form is: lower bound, upper bound, and textual parameters indicating the inclusion/
exclusion of bounds.
SELECT numrange(1.0, 14.0, '(]');
-- If the third parameter is ignored, it is assumed to be '[)'.
SELECT numrange(1.0, 14.0);
-- Although '(]' is specified here, the value will be converted to the standard format when displayed,
because int8range is a discrete range type (see below).
SELECT int8range(1, 14, '(]');
-- Using NULL for a bound causes the range to be unbounded on that side.
SELECT numrange(NULL, 2.2);
```

Discrete Range

A range element type has a well-defined "step" such as integer or date. In these types, if there is no valid value between two elements, they can be said to be adjacent. This is in contrast to a continuous range in which other element values can always be identified between two given values. For example, a range above the numeric type is continuous, and the range of timestamp is also continuous. (Although timestamp has limited precision and can be considered as discrete in theory, it is better to consider it as continuous because the step is not normally considered.)

Another way to consider discrete range types is to have a clear "next" or "previous" value for each element value. With this idea in mind, you can switch between inclusion and exclusion expressions of a range bound by replacing it with the original given next or previous element value. For example, in an integer range type, [4,8] and (3,9) represent the same set of values, but not for numeric ranges.

A discrete range type should have a *regularization* function that knows the expected step size of the element type. The regularization function can convert the equivalents of the range type to the same expression, in particular consistent with the inclusion or exclusion bounds. If you do not specify a regularization function, ranges with different formats will always be considered as unequal, even if they actually express the same set of values.

The built-in range types int4range, int8range, and daterange use a regularized form that includes the lower bound and excludes the upper bound, that is, [). However, user-defined range types can use other conventions.

Defining New Range

Users can define their own range types. The most common reason is to use the range on the subtype that is not provided in the built-in range type. For example, to create the range type subtype float8, run the following command:

```
CREATE TYPE floatrange AS RANGE (
    subtype = float8,
    subtype_diff = float8mi
);
SELECT '[1.234, 5.678]'::floatrange;
```

Because float8 does not have a meaningful "step", we did not define a regularization function in this example.

Defining your own range type also allows you to specify a different subtype B-tree operator class or collection to change the sort order to determine which values fall within the given range.

If the subtype is considered to have a discrete value instead of a continuous value, the **CREATE TYPE** command should specify a canonical function. The regularization function receives an input range value and must return an equivalent range value that may have different bounds and formats. For two ranges, for example, [1, 7] and [1, 8) that represent the same value set, the output must be the same. There is no relationship between choosing which expression to use as the regularization function, as long as two values of equal value in different formats can always be mapped to the same value in the same format. In addition to adjusting the inclusion/exclusion bound format, if the expected compensation is larger than the subtype can store, a regularization function may round the bound value. For example, a range type above a timestamp might be defined as having a one-hour epoch, so the regularization function might need to round off bounds that are not multiples of an hour, or might throw an error directly.

In addition, any range type intended to be used with a GiST or SP-GiST index should define a subtype difference or subtype_diff function (the index can still work without subtype_diff, but may be less efficient than when a difference function is provided). The subtype difference function uses two subtype input values and returns a difference expressed as a float8 value (X minus Y). In the example above, we can use functions under the regular float8 subtraction operator. However, for any other subtype, some type conversion may be required. There may also be a need for innovative ideas on how to express differences as numbers. For maximum extensibility, the subtype_diff function should agree with the sort order of the selected operator class and sort rules. That is, if the first parameter of the sort order is greater than the second parameter, the result should be a positive value.

The following is an example of a less simplified subtype_diff function:

```
CREATE FUNCTION time_subtype_diff(x time, y time) RETURNS float8 AS 'SELECT EXTRACT(EPOCH FROM
(x - y))' LANGUAGE sql STRICT IMMUTABLE;
CREATE TYPE timerange AS RANGE (
    subtype = time,
    subtype_diff = time_subtype_diff
 );
SELECT '[11:10, 23:00]'::timerange;
```

For details about how to create a range type, see **CREATE TYPE**.

Index

GiST and SP-GiST indexes can be created for table columns of the range type. For example, to create a GiST index, run the following command:

```
CREATE INDEX reservation_idx ON reservation USING GIST (during);
```

A GiST or SP-GiST index can accelerate queries involving the following range operators: =, &&, <@, @>, <<, >>, -|-, &<, and &> (see **Range Functions and Operators**).

In addition, the B-tree and hash index can be created on table columns of the range type. For these index types, basically the only useful range operation is equivalence. Using the corresponding < and > operators, there is a B-tree sort oder for range value definitions, but that order is fairly arbitrary and is often less useful in the reality. The B-tree and hash support for range types is primarily designed to allow sorting and hashing within a query, rather than creating an index.

11.3.15 OID Types

OIDs are used internally by GaussDB as primary keys for various system catalogs. OIDs are not added to user-created tables by the system. The OID type represents an object identifier.

The OID type is currently implemented as an unsigned four-byte integer. So, using a user-created table's **OID** column as a primary key is discouraged.

Table 11-22 OID types

Name	Reference	Description	Example
OID	N/A	Numeric object identifier	564182

Name	Reference	Description	Example
CID	N/A	Command identifier. This is the data type of the system columns **cmin** and **cmax**. Command identifiers are 32-bit quantities.	N/A
XID	N/A	Transaction identifier. This is the data type of the system columns **xmin** and **xmax**. Transaction identifiers are also 64-bit quantities.	N/A
TID	N/A	Row identifier. This is the data type of the system column **ctid**. A row ID is a pair (block number, tuple index within block) that identifies the physical location of the row within its table.	N/A
REGCONFIG	pg_ts_config	Text search configuration	english
REGDICTIONARY	pg_ts_dict	Text search dictionary	simple
REGOPER	pg_operator	Operator name	N/A
REGOPERATOR	pg_operator	Operator with parameter types	*(integer,integer) or -(NONE,integer)
REGPROC	pg_proc	Function name	sum
REGPROCEDURE	pg_proc	Function with parameter types	sum(int4)
REGCLASS	pg_class	Relation name	pg_type
REGTYPE	pg_type	Data type name	integer

The OID type is used for a column in the database system catalog.

Example:

```
openGauss=# SELECT oid FROM pg_class WHERE relname = 'pg_type';
 oid
------
 1247
(1 row)
```

The alias type for **OID** is **REGCLASS** which allows simplified search for **OID** values.

Example:

```
openGauss=# SELECT attrelid,attname,atttypid,attstattarget FROM pg_attribute WHERE attrelid =
'pg_type'::REGCLASS;
 attrelid |  attname   | atttypid | attstattarget
----------+------------+----------+---------------
     1247 | xc_node_id     |       23 |             0
     1247 | tableoid       |       26 |             0
     1247 | cmax           |       29 |             0
     1247 | xmax           |       28 |             0
     1247 | cmin           |       29 |             0
     1247 | xmin           |       28 |             0
     1247 | oid            |       26 |             0
     1247 | ctid           |       27 |             0
     1247 | typname        |       19 |            -1
     1247 | typnamespace   |       26 |            -1
     1247 | typowner       |       26 |            -1
     1247 | typlen         |       21 |            -1
     1247 | typbyval       |       16 |            -1
     1247 | typtype        |       18 |            -1
     1247 | typcategory    |       18 |            -1
     1247 | typispreferred |       16 |            -1
     1247 | typisdefined   |       16 |            -1
     1247 | typdelim       |       18 |            -1
     1247 | typrelid       |       26 |            -1
     1247 | typelem        |       26 |            -1
     1247 | typarray       |       26 |            -1
     1247 | typinput       |       24 |            -1
     1247 | typoutput      |       24 |            -1
     1247 | typreceive     |       24 |            -1
     1247 | typsend        |       24 |            -1
     1247 | typmodin       |       24 |            -1
     1247 | typmodout      |       24 |            -1
     1247 | typanalyze     |       24 |            -1
     1247 | typalign       |       18 |            -1
     1247 | typstorage     |       18 |            -1
     1247 | typnotnull     |       16 |            -1
     1247 | typbasetype    |       26 |            -1
     1247 | typtypmod      |       23 |            -1
     1247 | typndims       |       23 |            -1
     1247 | typcollation   |       26 |            -1
     1247 | typdefaultbin  |      194 |            -1
     1247 | typdefault     |       25 |            -1
     1247 | typacl         |     1034 |            -1
(38 rows)
```

11.3.16 Pseudo-Types

GaussDB type system contains a number of special-purpose entries that are collectively called pseudo-types. A pseudo-type cannot be used as a column data type, but it can be used to declare a function's argument or result type.

Each of the available pseudo-types is useful in situations where a function's behavior does not correspond to simply taking or returning a value of a specific SQL data type. **Table 11-23** lists all pseudo-types.

Table 11-23 Pseudo-types

Name	Description
any	Indicates that a function accepts any input data type.
anyelement	Indicates that a function accepts any data type.

Name	Description
anyarray	Indicates that a function accepts any array data type.
anynonarray	Indicates that a function accepts any non-array data type.
anyenum	Indicates that a function accepts any enum data type.
anyrange	Indicates that a function accepts any range data type.
cstring	Indicates that a function accepts or returns a null-terminated C string.
internal	Indicates that a function accepts or returns a server-internal data type.
language_handler	Indicates that a procedural language call handler is declared to return **language_handler**.
fdw_handler	Indicates that a foreign-data wrapper handler is declared to return **fdw_handler**.
record	Identifies a function returning an unspecified row type.
trigger	Indicates that a trigger function is declared to return **trigger**.
void	Indicates that a function returns no value.
opaque	Indicates an obsolete type name that formerly served all the above purposes.

Functions coded in C (whether built in or dynamically loaded) can be declared to accept or return any of these pseudo data types. It is up to the function author to ensure that the function will behave safely when a pseudo-type is used as an argument type.

Functions coded in procedural languages can use pseudo-types only as allowed by their implementation languages. At present the procedural languages all forbid use of a pseudo-type as argument type, and allow only **void** and **record** as a result type. Some also support polymorphic functions using the **anyelement**, **anyarray**, **anynonarray**, **anyenum**, and **anyrange** types.

The **internal** pseudo-type is used to declare functions that are meant only to be called internally by the database system, and not by direct invocation in an SQL query. If a function has at least one **internal**-type argument then it cannot be called from SQL. You are advised not to create any function that is declared to return internal unless it has at least one **internal** argument.

Example:

```
-- Create a table.
openGauss=# create table t1 (a int);

-- Insert two data records.
openGauss=# insert into t1 values(1),(2);

-- Create the showall() function.
```

```
openGauss=# CREATE OR REPLACE FUNCTION showall() RETURNS SETOF record
AS $$ SELECT count(*) from t1; $$
LANGUAGE SQL;

-- Invoke the showall() function.
openGauss=# SELECT showall();
 showall
---------
 (2)
(1 row)

-- Delete the function.
openGauss=# DROP FUNCTION showall();

-- Delete the table.
openGauss=# drop table t1;
```

11.3.17 Data Types Supported by Column-store Tables

Table 11-24 lists the data types supported by column-store tables.

Table 11-24 Data types supported by column-store tables

Category	Data Type	Length	Supported or Not
Numeric Types	smallint	2	Supported
	integer	4	Supported
	bigint	8	Supported
	decimal	-1	Supported
	numeric	-1	Supported
	real	4	Supported
	double precision	8	Supported
	smallserial	2	Supported
	serial	4	Supported
	bigserial	8	Supported
	largeserial	-1	Supported

Category	Data Type	Length	Supported or Not
Monetary Types	money	8	Supported
Character Types	character varying(n), varchar(n)	-1	Supported
	character(n), char(n)	n	Supported
	character, char	1	Supported
	text	-1	Supported
	nvarchar	-1	Supported
	nvarchar2	-1	Supported
	name	64	Not supported
Date/Time Types	timestamp with time zone	8	Supported
	timestamp without time zone	8	Supported
	date	4	Supported
	time without time zone	8	Supported
	time with time zone	12	Supported
	interval	16	Supported
big object	clob	-1	Supported
	blob	-1	Not supported
other types	...	...	Not supported

11.3.18 Data Type Used by the Ledger Database

The ledger database uses the hash16 data type to store row-level hash digests or table-level hash digests, and uses the hash32 data type to store global hash digests or history table verification hashes. (The current feature is a lab feature. Contact Huawei technical support before using it.)

Table 11-25 Hash type of the ledger database

Name	Description	Storage Space	Value Range
HASH16	Stored as an unsigned 64-bit integer	8 bytes	0 to +18446744073709551615
HASH32	Stored as a group of 16 unsigned integer elements	16 bytes	Value range of an unsigned integer array of 16 elements

The hash16 data type is used to store row-level or table-level hash digests in the ledger database. After obtaining the hash sequence of a 16-character hexadecimal string, the system invokes the **hash16in** function to convert the sequence into an unsigned 64-bit integer and stores the integer in a hash16 variable. For example:

```
Hexadecimal string: e697da2eaa3a775b; 64-bit unsigned integer: 16615989244166043483
Hexadecimal string: ffffffffffffffff; 64-bit unsigned integer: 18446744073709551615
```

The hash32 data type is used to store the global hash digest or history table verification hash in the ledger database. After obtaining the hash sequence of a 32-character hexadecimal string, the system invokes the **hash32in** function to convert the sequence to an array containing 16 unsigned integer elements. For example:

```
Hexadecimal string: 685847ed1fe38e18f6b0e2b18c00edee
Hash32 array: [104,88,71,237,31,227,142,24,246,176,226,177,140,0,237,238]
```

11.3.19 ACLItem

The ACLItem data type is used to store object permission information. Its internal implementation is of the int type and supports the '*user1*=*privs*/*user2*' format.

The aclitem[] data type is an array consisting of ACL items. The supported format is {*user1*=*privs1*/*user3*, *user2*=*privs2*/*user3*}.

In the preceding command, *user1*, *user2*, and *user3* indicate the existing users or roles in the database, and *privs* indicates the permissions supported by the database. For details, see **Table 13-48**.

Example:

```
openGauss=# create table table_acl (id int,priv aclitem,privs aclitem[]);
-- Create a data table table_acl that contains three fields of the int, aclitem, and aclitem[] types.
```

```
openGauss=# insert into table_acl values (1,'user1=arw/omm','{omm=d/user2,omm=w/omm}');
-- Insert a record whose content is (1,'user1=arw/omm','{omm=d/user2,omm=w/omm}') into the table_acl
table.
openGauss=# insert into table_acl values (2,'user1=aw/omm','{omm=d/user2}');
-- Insert a record whose content is (2,'user1=aw/omm','{omm=d/user2}') into the table_acl table.
openGauss=# select * from  table_acl;
 id |     priv      |          privs
----+---------------+-------------------------
  1 | user1=arw/omm | {omm=d/user2,omm=w/omm}
  2 | user1=aw/omm  | {omm=d/user2}
(2 rows)
```

11.4 Constant and Macro

Table 11-26 lists the constants and macros that can be used in GaussDB.

Table 11-26 Constant and macro

Parameter	Description	Example

CURRENT_CATALOG	Specifies the current database.	`openGauss=# SELECT CURRENT_CATALOG;`
		`current_database`
		`------------------`
		`openGauss`
		`(1 row)`

CURRENT_ROLE	Specifies the current user.	`openGauss=# SELECT CURRENT_ROLE;`
		`current_user`
		`--------------`
		`omm`
		`(1 row)`

CURRENT_SCHEMA	Specifies the current database schema.	`openGauss=# SELECT CURRENT_SCHEMA;`
		`current_schema`
		`----------------`
		`public`
		`(1 row)`

CURRENT_USER	Specifies the current user.	`openGauss=# SELECT CURRENT_USER;`
		`current_user`
		`--------------`
		`omm`
		`(1 row)`

LOCALTIMESTAMP	Specifies the current session time (without time zone).	`openGauss=# SELECT LOCALTIMESTAMP;`
		`         timestamp`
		`----------------------------`
		`2015-10-10 15:37:30.968538`
		`(1 row)`

NULL	This parameter is left blank.	N/A

SESSION_USER	Specifies the current system user.	`openGauss=# SELECT SESSION_USER;`
		`session_user`
		`--------------`
		`omm`
		`(1 row)`

SYSDATE	Specifies the current system date.	`openGauss=# SELECT SYSDATE;`
		`sysdate`
		`---------------------`
		`2015-10-10 15:48:53`
		`(1 row)`

Parameter	Description	Example

USER	Specifies the current user, also called **CURRENT_USER**.	`openGauss=# SELECT USER;`
		`current_user`
		`--------------`
		`omm`
		`(1 row)`

11.5 Functions and Operators

11.5.1 Logical Operators

The usual logical operators include AND, OR, and NOT. SQL uses a three-valued logical system with true, false, and null, which represents "unknown". Their priorities are NOT > AND > OR.

Table 11-27 lists the calculation rules, where a and b represent logical expressions.

Table 11-27 Operation rules

a	b	a AND b Result	a OR b Result	NOT a Result
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE
TRUE	NULL	NULL	TRUE	FALSE
FALSE	FALSE	FALSE	FALSE	TRUE
FALSE	NULL	FALSE	NULL	TRUE
NULL	NULL	NULL	NULL	NULL

NOTE

The operators AND and OR are commutative, that is, you can switch the left and right operand without affecting the result.

11.5.2 Comparison Operators

Comparison operators are available for the most data types and return Boolean values.

All comparison operators are binary operators. Only data types that are the same or can be implicitly converted can be compared using comparison operators.

Table 11-28 describes comparison operators provided by GaussDB.

Table 11-28 Comparison operators

Operator	Description
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
=	Equal to
<>, !=, or ^=	Not equal to

Comparison operators are available for all relevant data types. All comparison operators are binary operators that returned values of Boolean type. The calculation priority of the inequality sign is higher than that of the equality sign. If the entered data is different and cannot be implicitly converted, the comparison fails. For example, an expression such as 1<2<3 is invalid because the less-than sign (<) cannot be used to compare Boolean values and 3.

Besides, each comparison operator has a corresponding function in the **pg_proc** system catalog. If the value of proleakproof attribute of the corresponding function is **f**, the function is not used to prevent data leakage. If a user only has the permission for a system view, but does not have the permission for the corresponding table, the query plan may not be optimal when the user searches the system view.

11.5.3 Character Processing Functions and Operators

String functions and operators provided by GaussDB are for concatenating strings with each other, concatenating strings with non-strings, and matching the patterns of strings. Note: Except length-related functions, other functions and operators of string processing functions do not support parameters greater than 1 GB.

- bit_length(string)

  Description: Specifies the number of bits occupied by a string.

  Return type: int

  Example:

```
openGauss=# SELECT bit_length('world');
 bit_length
------------
         40
(1 row)
```

- btrim(string text [, characters text])

  Description: Removes the longest string consisting only of characters in **characters** (a space by default) from the start and end of **string**.

  Return type: text

  Example:

```
openGauss=# SELECT btrim('sring' , 'ing');
 btrim
```

```
-------
 sr
(1 row)
```

- char_length(string) or character_length(string)

  Description: Specifies the number of characters in a string.

  Return type: int

  Example:

```
openGauss=# SELECT char_length('hello');
 char_length
-------------
           5
(1 row)
```

- instr(text,text,int,int)

  Description: **instr(string1,string2,int1,int2)** returns the text from **int1** to **int2** in **string1**. The first **int** indicates the start position for matching, and the second **int** indicates the number of matching times.

  Return type: int

  Example:

```
openGauss=# SELECT instr( 'abcdabcdabcd', 'bcd', 2, 2 );
 instr
-------
     6
(1 row)
```

- lengthb(text/bpchar)

  Description: Obtains the number of bytes of a specified string.

  Return type: int

  Example:

```
openGauss=# SELECT lengthb('hello');
 lengthb
---------
       5
(1 row)
```

- left(str text, n int)

  Description: Returns the first *n* characters in a string. When *n* is negative, all but the last **|n|** characters are returned.

  Return type: text

  Example:

```
openGauss=# SELECT left('abcde', 2);
 left
------
 ab
(1 row)
```

- length(string bytea, encoding name)

  Description: Specifies the number of characters in **string** in the given **encoding**. **string** must be valid in this encoding.

  Return type: int

  Example:

```
openGauss=# SELECT length('jose', 'UTF8');
 length
--------
      4
(1 row)
```

NOTE

If the length of the bytea type is queried and UTF8 encoding is specified, the maximum length can only be **536870888**.

- lpad(string text, length int [, fill text])

  Description: Fills up **string** to **length** by appending the characters **fill** (a space by default). If **string** is already longer than **length**, then it is truncated.

  Return type: text

  Example:

```
openGauss=# SELECT lpad('hi', 5, 'xyza');
 lpad
-------
 xyzhi
(1 row)
```

- notlike(x bytea name text, y bytea text)

  Description: Compares x and y to check whether they are inconsistent.

  Return type: Boolean

  Example:

```
openGauss=# SELECT notlike(1,2);
   notlike
--------------
          t
(1 row)
openGauss=# SELECT notlike(1,1);
   notlike
--------------
          f
(1 row)
```

- octet_length(string)

  Description: Specifies the number of bytes in a string.

  Return type: int

  Example:

```
openGauss=# SELECT octet_length('jose');
 octet_length
--------------
          4
(1 row)
```

- overlay(string placing string FROM int [for int])

  Description: Replaces substrings. **FROM int** indicates the start position of the replacement in the first string. **for int** indicates the number of characters replaced in the first string.

  Return type: text

  Example:

```
openGauss=# SELECT overlay('hello' placing 'world' from 2 for 3 );
 overlay
---------
 hworldo
(1 row)
```

- position(substring in string)

  Description: Specifies the position of a substring. Parameters are case-sensitive.

  Return type: int. If the character string does not exist, **0** is returned.

Example:

```
openGauss=# SELECT position('ing' in 'string');
 position
----------
        4
(1 row)
```

- pg_client_encoding()

  Description: Specifies the current client encoding name.

  Return type: name

  Example:

```
openGauss=# SELECT pg_client_encoding();
 pg_client_encoding
--------------------
 UTF8
(1 row)
```

- quote_ident(string text)

  Description: Returns the given string suitably quoted to be used as an identifier in an SQL statement string (quotation marks are used as required). Quotation marks are added only if necessary (that is, if the string contains non-identifier characters or would be case-folded). Embedded quotation marks are properly doubled.

  Return type: text

  Example:

```
openGauss=# SELECT quote_ident('hello world');
 quote_ident
--------------
 "hello world"
(1 row)
```

- quote_literal(string text)

  Description: Returns the given string suitably quoted to be used as a string literal in an SQL statement string (quotation marks are used as required).

  Return type: text

  Example:

```
openGauss=# SELECT quote_literal('hello');
 quote_literal
---------------
 'hello'
(1 row)
```

  If a command similar to the following exists, the text will be escaped.

```
openGauss=# SELECT quote_literal(E'O\'hello');
 quote_literal
---------------
 'O''hello'
(1 row)
```

  If a command similar to the following exists, the backslash will be properly doubled.

```
openGauss=# SELECT quote_literal('O\hello');
 quote_literal
---------------
 E'O\\hello'
(1 row)
```

  If the parameter is null, **NULL** is returned. If the parameter may be null, you are advised to use **quote_nullable**.

```
openGauss=# SELECT quote_literal(NULL);
 quote_literal
---------------

(1 row)
```

- quote_literal(value anyelement)

  Description: Converts the given value to text and then quotes it as a literal.

  Return type: text

  Example:

  ```
  openGauss=# SELECT quote_literal(42.5);
   quote_literal
  ---------------
   '42.5'
  (1 row)
  ```

  If a command similar to the following exists, the given value will be escaped.

  ```
  openGauss=# SELECT quote_literal(E'O\'42.5');
   quote_literal
  ---------------
   '0''42.5'
  (1 row)
  ```

  If a command similar to the following exists, the backslash will be properly doubled.

  ```
  openGauss=# SELECT quote_literal('O\42.5');
   quote_literal
  ---------------
   E'O\\42.5'
  (1 row)
  ```

- quote_nullable(string text)

  Description: Returns the given string suitably quoted to be used as a string literal in an SQL statement string (quotation marks are used as required).

  Return type: text

  Example:

  ```
  openGauss=# SELECT quote_nullable('hello');
   quote_nullable
  ----------------
   'hello'
  (1 row)
  ```

  If a command similar to the following exists, the text will be escaped.

  ```
  openGauss=# SELECT quote_nullable(E'O\'hello');
   quote_nullable
  ----------------
   'O''hello'
  (1 row)
  ```

  If a command similar to the following exists, the backslash will be properly doubled.

  ```
  openGauss=# SELECT quote_nullable('O\hello');
   quote_nullable
  ----------------
   E'O\\hello'
  (1 row)
  ```

  If the parameter is null, **NULL** is returned.

  ```
  openGauss=# SELECT quote_nullable(NULL);
   quote_nullable
  ----------------
   NULL
  (1 row)
  ```

- quote_nullable(value anyelement)

  Description: Converts the given value to text and then quotes it as a literal.

  Return type: text

  Example:

```
openGauss=# SELECT quote_nullable(42.5);
 quote_nullable
----------------
 '42.5'
(1 row)
```

  If a command similar to the following exists, the given value will be escaped.

```
openGauss=# SELECT quote_nullable(E'O\'42.5');
 quote_nullable
----------------
 'O''42.5'
(1 row)
```

  If a command similar to the following exists, the backslash will be properly doubled.

```
openGauss=# SELECT quote_nullable('O\42.5');
 quote_nullable
----------------
 E'O\\42.5'
(1 row)
```

  If the parameter is null, **NULL** is returned.

```
openGauss=# SELECT quote_nullable(NULL);
 quote_nullable
----------------
 NULL
(1 row)
```

- substring_inner(string [from int] [for int])

  Description: Extracts a substring. **from int** indicates the start position of the truncation. **for int** indicates the number of characters truncated.

  Return type: text

  Example:

```
openGauss=# select substring_inner('adcde', 2,3);
 substring_inner
-----------------
 dcd
(1 row)
```

- substring(string [from int] [for int])

  Description: Extracts a substring. **from int** indicates the start position of the truncation. **for int** indicates the number of characters truncated.

  Return type: text

  Example:

```
openGauss=# SELECT substring('Thomas' from 2 for 3);
 substring
-----------
 hom
(1 row)
```

- substring(string from *pattern*)

  Description: Extracts substrings matching the POSIX-style regular expression. It returns the text that matches the pattern. If no match record is found, a null value is returned.

  Return type: text

Example:

```
openGauss=# SELECT substring('Thomas' from '...$');
 substring
-----------
 mas
(1 row)
openGauss=# SELECT substring('foobar' from 'o(.)b');
 result
--------
 o
(1 row)
openGauss=# SELECT substring('foobar' from '(o(.)b)');
 result
--------
 oob
(1 row)
```

NOTE

If the POSIX-style regular expression contains any parentheses, the portion of the text that matched the first parenthesized sub-expression (the one whose left parenthesis comes first) is returned. You can put parentheses around the whole expression if you want to use parentheses within it without triggering this exception.

- substring(string from *pattern* for *escape*)

  Description: Extracts substrings matching the SQL regular expression. The declared schema must match the entire data string; otherwise, the function fails and returns a null value. To indicate the part of the pattern that should be returned on success, the pattern must contain two occurrences of the escape character followed by a double quotation mark ("). The text matching the portion of the pattern between these marks is returned.

  Return type: text

  Example:

```
openGauss=# SELECT substring('Thomas' from '%#"o_a#"_' for '#');
 substring
-----------
 oma
(1 row)
```

- rawcat(raw,raw)

  Description: Indicates the string concatenation function.

  Return type: raw

  Example:

```
openGauss=# SELECT rawcat('ab','cd');
 rawcat
--------
 ABCD
(1 row)
```

- regexp_like(text,text,text)

  Description: Indicates the mode matching function of a regular expression.

  Return type: Boolean

  Example:

```
openGauss=# SELECT regexp_like('str','[ac]');
 regexp_like
-------------
 f
(1 row)
```

- regexp_substr(string text, pattern text [, position int [, occurrence int [, flags text]]])

Description: Extracts substrings from a regular expression. Its function is similar to **substr**. When a regular expression contains multiple parallel brackets, it also needs to be processed.

Parameter description:

- **string**: source character string used for matching.
- **pattern**: regular expression pattern string used for matching.
- **position**: start character of the source string used for matching. This parameter is optional. The default value is **1**.
- **occurrence**: sequence number of the matched substring to be extracted. This parameter is optional. The default value is **1**.
- **flags**: contains zero or multiple single-letter flags that change the matching behavior of the function. This parameter is optional. **m** indicates multi-line matching. If the SQL syntax is compatible with products A and B and the value of the GUC parameter **behavior_compat_options** contains **aformat_regexp_match**, the option **n** indicates that the period (.) can match the **'\n'** character. If **n** is not specified in flags, the period (.) cannot match the **'\n'** character by default. If the value does not contain **aformat_regexp_match**, the period (.) matches the **'\n'** character by default. The meaning of option **n** is the same as that of option **m**.

Return type: text

Example:

```
openGauss=# SELECT regexp_substr('str','[ac]');
 regexp_substr
---------------

(1 row)

openGauss=# SELECT regexp_substr('foobarbaz', 'b(..)', 3, 2) AS RESULT;
 result
--------
 baz
(1 row)
```

- regexp_count(string text, pattern text [, position int [, flags text]])

  Description: obtains the number of substrings used for matching.

  Parameter description:

  - **string**: source character string used for matching.
  - **pattern**: regular expression pattern string used for matching.
  - **position**: sequence number of the character to be matched from the source character string. This parameter is optional. The default value is **1**.
  - **flags**: contains zero or multiple single-letter flags that change the matching behavior of the function. This parameter is optional. **m** indicates multi-line matching. If the SQL syntax is compatible with products A and B and the value of the GUC parameter **behavior_compat_options** contains **aformat_regexp_match**, the option **n** indicates that the period (.) can match the **'\n'** character. If **n** is not specified in flags, the period (.) cannot match the **'\n'** character by default. If the value does not contain **aformat_regexp_match**, the period (.) matches the **'\n'** character by default. The meaning of option **n** is the same as that of option **m**.

  Return type: int

Example:

```
openGauss=# SELECT regexp_count('foobarbaz','b(..)', 5) AS RESULT;
result
--------
1
(1 row)
```

- regexp_instr(string text, pattern text [, position int [, occurrence int [, return_opt int [, flags text]]]])

  Description: obtains the position (starting from 1) of the substring that meets the matching condition. If no substring is matched, **0** is returned.

  Parameter description:

  – **string**: source character string used for matching.
  – **pattern**: regular expression pattern string used for matching.
  – **position**: start character of the source string used for matching. This parameter is optional. The default value is **1**.
  – **occurrence**: sequence number of the matched substring to be replaced. This parameter is optional. The default value is **1**.
  – **return_opt**: specifies whether to return the position of the first or last character of the matched substring. This parameter is optional. If the value is **0**, the position of the first character (starting from 1) of the matched substring is returned. If the value is greater than 0, the position of the next character of the end character of the matched substring is returned. The default value is **0**.
  – **flags**: contains zero or multiple single-letter flags that change the matching behavior of the function. This parameter is optional. **m** indicates multi-line matching. If the SQL syntax is compatible with products A and B and the value of the GUC parameter **behavior_compat_options** contains **aformat_regexp_match**, the option **n** indicates that the period (.) can match the **'\n'** character. If **n** is not specified in flags, the period (.) cannot match the **'\n'** character by default. If the value does not contain **aformat_regexp_match**, the period (.) matches the **'\n'** character by default. The meaning of option **n** is the same as that of option **m**.

  Return type: int

  Example:

```
openGauss=# SELECT regexp_instr('foobarbaz','b(..)', 1, 1, 0) AS RESULT;
result
--------
4
(1 row)

openGauss=# SELECT regexp_instr('foobarbaz','b(..)', 1, 2, 0) AS RESULT;
result
--------
7
(1 row)
```

- regexp_matches(string text, pattern text [, flags text])

  Description: Returns all captured substrings resulting from matching a POSIX-style regular expression against **string**. If the pattern does not match, the function returns no rows. If the pattern contains no parenthesized sub-expressions, then each row returned is a single-element text array containing the substring matching the whole pattern. If the pattern contains parenthesized sub-expressions, the function returns a text array whose *n*th element is the substring matching the *n*th parenthesized sub-expression of the pattern.

The optional **flags** argument contains zero or multiple single-letter flags that change the function behavior. **i** indicates that the matching is not related to uppercase and lowercase. **g** indicates that each matched substring is replaced, instead of replacing only the first one.

NOTICE

If the last parameter is provided but the parameter value is an empty string ('') and the SQL compatibility mode of the database is set to A, the returned result is an empty set. This is because the A compatibility mode treats the empty string ('') as **NULL**. To resolve this problem, you can:

- Change the database SQL compatibility mode to C.
- Do not provide the last parameter or do not set the last parameter to an empty string.

Return type: SETOF text[]

Example:

```
openGauss=# SELECT regexp_matches('foobarbequebaz', '(bar)(beque)');
 regexp_matches
----------------
 {bar,beque}
(1 row)
openGauss=# SELECT regexp_matches('foobarbequebaz', 'barbeque');
 regexp_matches
----------------
 {barbeque}
(1 row)
openGauss=#  SELECT regexp_matches('foobarbequebazilbarfbonk', '(b[^b]+)(b[^b]+)', 'g');
    result
--------------
 {bar,beque}
 {bazil,barf}
(2 rows)
```

- regexp_split_to_array(string text, pattern text [, flags text])

  Description: Splits **string** using a POSIX-style regular expression as the delimiter. The **regexp_split_to_array** function behaves the same as **regexp_split_to_table**, except that **regexp_split_to_array** returns its result as an array of text.

  Return type: text[]

  Example:

```
openGauss=# SELECT regexp_split_to_array('hello world', E'\\s+');
 regexp_split_to_array
-----------------------
 {hello,world}
(1 row)
```

- regexp_split_to_table(string text, pattern text [, flags text])

  Description: Splits **string** using a POSIX-style regular expression as the delimiter. If there is no match to the pattern, the function returns the string. If there is at least one match, for each match it returns the text from the end of the last match (or the beginning of the string) to the beginning of the match.

When there are no more matches, it returns the text from the end of the last match to the end of the string.

The **flags** parameter is a text string containing zero or more single-letter flags that change the function's behavior. **i** indicates case-insensitive matching.

Return type: SETOF text

Example:

```
openGauss=# SELECT regexp_split_to_table('hello world', E'\\s+');
 regexp_split_to_table
-----------------------
 hello
 world
(2 rows)
```

- repeat(string text, number int)

  Description: Repeats **string** the specified number of times.

  Return type: text

  Example:

```
openGauss=# SELECT repeat('Pg', 4);
  repeat
----------
 PgPgPgPg
(1 row)
```

  NOTE

  The maximum size of memory allocated at a time cannot exceed 1 GB due to the memory allocation mechanism of the database. Therefore, the maximum value of **number** cannot exceed (1 GB – **x**)/**lengthb** (**string**) – 1. **x** indicates the length of the header information, which is usually greater than 4 bytes. The value varies among different scenarios.

- replace(string text, from text, to text)

  Description: Replaces all occurrences in **string** of substring **from** with substring **to**.

  Return type: text

  Example:

```
openGauss=# SELECT replace('abcdefabcdef', 'cd', 'XXX');
    replace
----------------
 abXXXefabXXXef
(1 row)
```

- replace(string, substring)

  Description: Deletes all substrings in a string.

  String type: text

  Substring type: text

  Return type: text

  Example:

```
openGauss=# SELECT replace('abcdefabcdef', 'cd');
    replace
----------------
 abefabef
(1 row)
```

- reverse(str)

  Description: Returns the reversed string.

Return type: text

Example:

```
openGauss=# SELECT reverse('abcde');
 reverse
---------
 edcba
(1 row)
```

- right(str text, n int)

  Description: Returns the last *n* characters in a string. When *n* is negative, all but the first **|n|** characters are returned.

  Return type: text

  Example:

```
openGauss=# SELECT right('abcde', 2);
 right
-------
 de
(1 row)

openGauss=# SELECT right('abcde', -2);
 right
-------
 cde
(1 row)
```

- rpad(string text, length int [, fill text])

  Description: Fills up **string** to **length** by appending the characters **fill** (a space by default). If **string** is already longer than **length**, then it is truncated.

  Return type: text

  Example:

```
openGauss=# SELECT rpad('hi', 5, 'xy');
 rpad
-------
 hixyx
(1 row)
```

- rtrim(string text [, characters text])

  Description: Removes the longest string containing only characters from characters (a space by default) from the end of string.

  Return type: text

  Example:

```
openGauss=# SELECT rtrim('trimxxxx', 'x');
 rtrim
-------
 trim
(1 row)
```

- substrb(text,int,int)

  Description: Extracts a substring. The first **int** indicates the start position of the subtraction. The second **int** indicates the number of characters extracted.

  Return type: text

  Example:

```
openGauss=# SELECT substrb('string',2,3);
 substrb
---------
 tri
(1 row)
```

- substrb(text,int)

  Description: Extracts a substring. **int** indicates the start position of the extraction.

  Return type: text

  Example:

```
openGauss=# SELECT substrb('string',2);
 substrb
---------
 tring
(1 row)
```

- substr(bytea,from,count)

  Description: Extracts a substring from **bytea**. **from** specifies the position where the extraction starts. **count** specifies the length of the extracted substring.

  Return type: text

  Example:

```
openGauss=# SELECT substr('string',2,3);
 substr
--------
 tri
(1 row)
```

- string || string

  Description: Concatenates strings.

  Return type: text

  Example:

```
openGauss=# SELECT 'MPP'||'DB' AS RESULT;
 result
--------
 MPPDB
(1 row)
```

- string || non-string or non-string || string

  Description: Concatenates strings and non-strings.

  Return type: text

  Example:

```
openGauss=# SELECT 'Value: '||42 AS RESULT;
  result
-----------
 Value: 42
(1 row)
```

- split_part(string text, delimiter text, field int)

  Description: Splits **string** on **delimiter** and returns the **field**th column (counting from text of the first appeared delimiter).

  Return type: text

  Example:

```
openGauss=# SELECT split_part('abc~@~def~@~ghi', '~@~', 2);
 split_part
------------
 def
(1 row)
```

- strpos(string, substring)

Description: Specifies the position of a substring. It is the same as **position(substring in string)**. However, the parameter sequences of them are reversed.

Return type: int

Example:

```
openGauss=# SELECT strpos('source', 'rc');
 strpos
--------
      4
(1 row)
```

- to_hex(number int or bigint)

  Description: Converts a number to a hexadecimal expression.

  Return type: text

  Example:

```
openGauss=# SELECT to_hex(2147483647);
  to_hex
----------
 7fffffff
(1 row)
```

- translate(string text, from text, to text)

  Description: Any character in **string** that matches a character in the **from** set is replaced by the corresponding character in the **to** set. If **from** is longer than **to**, extra characters occurred in **from** are removed.

  Return type: text

  Example:

```
openGauss=# SELECT translate('12345', '143', 'ax');
 translate
-----------
 a2x5
(1 row)
```

- length(string)

  Description: Obtains the number of characters in a string.

  Return type: integer

  Example:

```
openGauss=# SELECT length('abcd');
 length
--------
      4
(1 row)
```

- lengthb(string)

  Description: Obtains the number of characters in a string. The value depends on character sets (GBK and UTF8).

  Return type: integer

  Example:

```
openGauss=# SELECT lengthb('Chinese');
 lengthb
---------
       7
(1 row)
```

- substr(string,from)

  Description:

Extracts substrings from a string.

from indicates the start position of the extraction.

- If **from** starts at 0, the value **1** is used.
- If the value of **from** is positive, all characters from **from** to the end are extracted.
- If the value of **from** is negative, the last *n* characters in the string are extracted, in which **n** indicates the absolute value of **from**.

Return type: varchar

Example:

If the value of **from** is positive:

```
openGauss=# SELECT substr('ABCDEF',2);
 substr
--------
 BCDEF
(1 row)
```

If the value of **from** is negative:

```
openGauss=# SELECT substr('ABCDEF',-2);
 substr
--------
 EF
(1 row)
```

- substr(string,from,count)

  Description:

  Extracts substrings from a string.

  from indicates the start position of the extraction.

  count indicates the length of the extracted substring.

  - If **from** starts at 0, the value **1** is used.
  - If the value of **from** is positive, extract **count** characters starting from **from**.
  - If the value of **from** is negative, extract the last **n count** characters in the string, in which **n** indicates the absolute value of **from**.
  - If the value of **count** is smaller than **1**, **null** is returned.

  Return type: varchar

  Example:

  If the value of **from** is positive:

```
openGauss=# SELECT substr('ABCDEF',2,2);
 substr
--------
 BC
(1 row)
```

  If the value of **from** is negative:

```
openGauss=# SELECT substr('ABCDEF',-3,2);
 substr
--------
 DE
(1 row)
```

- substrb(string,from)

  Description: The functionality of this function is the same as that of **SUBSTR(string,from)**. However, the calculation unit is byte.

Return type: bytea

Example:

```
openGauss=# SELECT substrb('ABCDEF',-2);
 substrb
---------
 EF
(1 row)
```

- substrb(string,from,count)

  Description: The functionality of this function is the same as that of **SUBSTR(string,from,count)**. However, the calculation unit is byte.

  Return type: bytea

  Example:

```
openGauss=# SELECT substrb('ABCDEF',2,2);
 substrb
---------
 BC
(1 row)
```

- trim([leading |trailing |both] [characters] from string)

  Description: Removes the longest string containing only the characters (a space by default) from the start/end/both ends of the string.

  Return type: text

  Example:

```
openGauss=# SELECT trim(BOTH 'x' FROM 'xTomxx');
 btrim
-------
 Tom
(1 row)
openGauss=# SELECT trim(LEADING 'x' FROM 'xTomxx');
 ltrim
-------
 Tomxx
(1 row)
openGauss=# SELECT trim(TRAILING 'x' FROM 'xTomxx');
 rtrim
-------
 xTom
(1 row)
```

- rtrim(string [, characters])

  Description: Removes the longest string containing only characters from characters (a space by default) from the end of string.

  Return type: text

  Example:

```
openGauss=# SELECT rtrim('TRIMxxxx','x');
 rtrim
-------
 TRIM
(1 row)
```

- ltrim(string [, characters])

  Description: Removes the longest string containing only characters from characters (a space by default) from the start of string.

  Return type: text

  Example:

```
openGauss=# SELECT ltrim('xxxxTRIM','x');
 ltrim
```

```
-------
 TRIM
(1 row)
```

- upper(string)

  Description: Converts the string into the uppercase.

  Return type: text

  Example:

  ```
  openGauss=# SELECT upper('tom');
   upper
  -------
   TOM
  (1 row)
  ```

- lower(string)

  Description: Converts the string into the lowercase.

  Return type: text

  Example:

  ```
  openGauss=# SELECT lower('TOM');
   lower
  -------
   tom
  (1 row)
  ```

- rpad(string varchar, length int [, fill varchar])

  Description: Fills up **string** to **length** by appending the characters **fill** (a space by default). If **string** is already longer than **length**, then it is truncated.

  length in GaussDB indicates the character length. One Chinese character is counted as one character.

  Return type: text

  Example:

  ```
  openGauss=# SELECT rpad('hi',5,'xyza');
   rpad
  -------
   hixyz
  (1 row)
  openGauss=# SELECT rpad('hi',5,'abcdefg');
   rpad
  -------
   hiabc
  (1 row)
  ```

- instr(string,substring[,position,occurrence])

  Description: Queries and returns the value of the substring position that occurs the **occurrence** (1 by default) times from the **position** (1 by default) in the string.

  – If the value of **position** is **0**, **0** is returned.
  – If the value of **position** is negative, the search is performed backwards from the last *n*th character in the string, in which *n* indicates the absolute value of **position**.

  In this function, the calculation unit is character. One Chinese character is one character.

  Return type: integer

  Example:

  ```
  openGauss=# SELECT instr('corporate floor','or', 3);
   instr
  ```

```
-------
     5
(1 row)
openGauss=# SELECT instr('corporate floor','or',-3,2);
 instr
-------
     2
(1 row)
```

- initcap(string)

  Description: Converts the first letter of each word in the string into the uppercase and the other letters into the lowercase.

  Return type: text

  Example:

```
openGauss=# SELECT initcap('hi THOMAS');
  initcap
-----------
 Hi Thomas
(1 row)
```

- ascii(string)

  Description: Indicates the ASCII code of the first character in the string.

  Return type: integer

  Example:

```
openGauss=# SELECT ascii('xyz');
 ascii
-------
   120
(1 row)
```

- replace(string varchar, search_string varchar, replacement_string varchar)

  Description: Replaces all **search_string** in the string with **replacement_string**.

  Return type: varchar

  Example:

```
openGauss=# SELECT replace('jack and jue','j','bl');
    replace
----------------
 black and blue
(1 row)
```

- lpad(string varchar, length int[, repeat_string varchar])

  Description: Adds a series of **repeat_string** (a space by default) on the left of the string to generate a new string with the total length of *n*.

  If the length of the string is longer than the specified length, the function truncates the string and returns the substrings with the specified length.

  Return type: varchar

  Example:

```
openGauss=# SELECT lpad('PAGE 1',15,'*.');
      lpad
-----------------
 *.*.*.*.*PAGE 1
(1 row)
openGauss=# SELECT lpad('hello world',5,'abcd');
 lpad
-------
 hello
(1 row)
```

- concat(str1,str2)

Description: Connects str1 and str2 and returns the string. Note: **concat** calls the output function of the data type and the return value is immutable. As a result, the optimizer cannot calculate the result in advance when generating a plan. If there are performance requirements, you are advised to use the operator **||**.

NOTICE

If the SQL compatibility mode is set to **MY** and **str1** or **str2** is set to **NULL**, **NULL** will be returned.

Return type: varchar

Example:

```
openGauss=# SELECT concat('Hello', ' World!');
    concat
--------------
 Hello World!
(1 row)
openGauss=# SELECT concat('Hello', NULL);
 concat
--------
 Hello
(1 row)
```

- chr(integer)

  Description: Specifies the character of the ASCII code.

  Return type: varchar

  Example:

```
openGauss=# SELECT chr(65);
 chr
-----
 A
(1 row)
```

- regexp_substr(source_char, pattern)

  Description: Extracts substrings from a regular expression. If the SQL syntax is compatible with products A and B and the value of the GUC parameter **behavior_compat_options** contains **aformat_regexp_match**, the period (.) cannot match the **'\n'** character. If **aformat_regexp_match** is not contained, the period (.) matches the **'\n'** character by default.

  Return type: text

  Example:

```
openGauss=# SELECT regexp_substr('500 Hello World, Redwood Shores, CA', ',[^,]+,')
"REGEXPR_SUBSTR";
  REGEXPR_SUBSTR
-------------------
 , Redwood Shores,
(1 row)
```

- regexp_replace(string, pattern, replacement [,flags])

  Description: Replaces substrings matching the POSIX-style regular expression. The source string is returned unchanged if there is no match to the pattern. If there is a match, the source string is returned with the replacement string substituted for the matching substring.

  The replacement string can contain **\n**, where **n** is 1 through 9, to indicate that the source substring matching the *n*th parenthesized sub-expression of the pattern should be inserted, and it can contain **\&** to indicate that the substring matching the entire pattern should be inserted.

The optional **flags** argument contains zero or multiple single-letter flags that change the function behavior. **i** indicates that the matching is not related to uppercase and lowercase. **g** indicates that each matched substring is replaced, instead of replacing only the first one. **m** indicates multi-line matching. If the SQL syntax is compatible with products A and B and the value of the GUC parameter **behavior_compat_options** contains **aformat_regexp_match**, the option **n** indicates that the period (.) can match the **'\n'** character. If **n** is not specified in flags, the period (.) cannot match the **'\n'** character by default. If the value does not contain **aformat_regexp_match**, the period (.) matches the **'\n'** character by default. The meaning of option **n** is the same as that of option **m**.

Return type: varchar

Example:

```
openGauss=# SELECT regexp_replace('Thomas', '.[mN]a.', 'M');
 regexp_replace
----------------
 ThM
(1 row)
openGauss=# SELECT regexp_replace('foobarbaz','b(..)', E'X\\1Y', 'g') AS
RESULT;
   result
-------------
 fooXarYXazY
(1 row)
```

- repexp_replace(string text, pattern text [, replacement text [, position int [, occurrence int [, flags text]]]])

  Description: Replaces substrings matching the POSIX-style regular expression. The source string is returned unchanged if there is no match to the pattern. If there is a match, the source string is returned with the replacement string substituted for the matching substring.

  Parameter description:

  – **string**: source character string used for matching.

  – **pattern**: regular expression pattern string used for matching.

  – **replacement**:character string used to replace the matched substring. This parameter is optional. If no parameter value is specified or the parameter value is null, the parameter value is replaced with an empty string.

  – **position**: start character of the source string used for matching. This parameter is optional. The default value is **1**.

  – **occurrence**: sequence number of the matched substring to be replaced. This parameter is optional. The default value is **0**, indicating that all matched substrings are replaced.

  – **flags**: contains zero or multiple single-letter flags that change the matching behavior of the function. This parameter is optional. **m** indicates multi-line matching. If the SQL syntax is compatible with products A and B and the value of the GUC parameter **behavior_compat_options** contains **aformat_regexp_match**, the option **n** indicates that the period (.) can match the **'\n'** character. If **n** is not specified in flags, the period (.) cannot match the **'\n'** character by default. If the value does not contain **aformat_regexp_match**, the period

(.) matches the **'\n'** character by default. The meaning of option **n** is the same as that of option **m**.

Return type: text

Example:

```
openGauss=# SELECT regexp_replace('foobarbaz','b(..)', E'X\\1Y', 2, 2, 'n') AS RESULT;
result
------------
foobarXazY
(1 row)
```

- concat_ws(sep text, str"any" [, str"any" [, ...]])

  Description: Uses the first parameter as the separator, which is associated with all following parameters. The **NULL** parameter is ignored.

  NOTICE

  - If the first parameter value is **NULL**, the returned result is **NULL**.
  - If the first parameter is provided but the parameter value is an empty string ('') and the SQL compatibility mode of the database is set to **A**, the returned result is **NULL**. This is because the A compatibility mode treats the empty string ('') as **NULL**. To resolve this problem, you can change the SQL compatibility mode of the database to **B**, **C**, or **PG**.

  Return type: text

  Example:

```
openGauss=# SELECT concat_ws(',', 'ABCDE', 2, NULL, 22);
 concat_ws
------------
 ABCDE,2,22
(1 row)
```

- nlssort(string text, sort_method text)

  Description: Returns the encoding value of a string in the sorting mode specified by **sort_method**. The encoding value can be used for sorting and determines the sequence of the string in the sorting mode. Currently, **sort_method** can be set to **nls_sort=schinese_pinyin_m** or **nls_sort=generic_m_ci**. **nls_sort=generic_m_ci** supports only the case-insensitive order for English characters.

  String type: text

  sort_method type: text

  Return type: text

  Example:

```
openGauss=# SELECT nlssort('A', 'nls_sort=schinese_pinyin_m');
    nlssort
----------------
 01EA0000020006
(1 row)
openGauss=# SELECT nlssort('A', 'nls_sort=generic_m_ci');
    nlssort
----------------
 01EA000002
(1 row)
```

- convert(string bytea, src_encoding name, dest_encoding name)

Description: Converts the bytea string to **dest_encoding**. **src_encoding** specifies the source code encoding. The string must be valid in this encoding.

Return type: bytea

Example:

```
openGauss=# SELECT convert('text_in_utf8', 'UTF8', 'GBK');
          convert
----------------------------
 \x746578745f696e5f75746638
(1 row)
```

NOTE

If the rule for converting between source to target encoding (for example, GBK and LATIN1) does not exist, the string is returned without conversion. See the **pg_conversion** system catalog for details.

Example:

```
openGauss=# show server_encoding;
 server_encoding
-----------------
 LATIN1
(1 row)

openGauss=# SELECT convert_from('some text', 'GBK');
 convert_from
--------------
 some text
(1 row)

db_latin1=# SELECT convert_to('some text', 'GBK');
      convert_to
----------------------
 \x736f6d652074657874
(1 row)

db_latin1=# SELECT convert('some text', 'GBK', 'LATIN1');
       convert
----------------------
 \x736f6d652074657874
(1 row)
```

- convert_from(string bytea, src_encoding name)

  Description: Converts the long bytea using the coding mode of the database.

  src_encoding specifies the source code encoding. The string must be valid in this encoding.

  Return type: text

  Example:

```
openGauss=# SELECT convert_from('text_in_utf8', 'UTF8');
 convert_from
--------------
 text_in_utf8
(1 row)
```

- convert_to(string text, dest_encoding name)

  Description: Converts a string to **dest_encoding**.

  Return type: bytea

  Example:

```
openGauss=# SELECT convert_to('some text', 'UTF8');
      convert_to
----------------------
 \x736f6d652074657874
(1 row)
```

- string [NOT] LIKE pattern [ESCAPE escape-character]

  Description: Specifies the pattern matching function.

  If the pattern does not include a percentage sign (%) or an underscore (_), this mode represents itself only. In this case, the behavior of LIKE is the same as the equal operator. The underscore (_) in the pattern matches any single character while one percentage sign (%) matches no or multiple characters.

  To match with underscores (_) or percent signs (%), corresponding characters in **pattern** must lead escape characters. The default escape character is a backward slash (\) and can be specified using the **ESCAPE** clause. To match with escape characters, enter two escape characters.

  Return type: Boolean

  Example:

```
openGauss=# SELECT 'AA_BBCC' LIKE '%A@_B%' ESCAPE '@' AS RESULT;
 result
--------
 t
(1 row)
openGauss=# SELECT 'AA_BBCC' LIKE '%A@_B%' AS RESULT;
 result
--------
 f
(1 row)
openGauss=# SELECT 'AA@_BBCC' LIKE '%A@_B%' AS RESULT;
 result
--------
 t
(1 row)
```

- REGEXP_LIKE(source_string, pattern [, match_parameter])

  Description: Indicates the mode matching function of a regular expression.

  source_string indicates the source string and **pattern** indicates the matching pattern of the regular expression. **match_parameter** indicates the matching items and the values are as follows:

  – 'i': case-insensitive

  – 'c': case-sensitive

  – 'n': allowing the metacharacter "." in a regular expression to be matched with a linefeed.

  – 'm': allows **source_string** to be regarded as multiple rows.

  If **match_parameter** is ignored, **case-sensitive** is enabled by default, "." is not matched with a linefeed, and **source_string** is regarded as a single row.

  Return type: Boolean

  Example:

```
openGauss=# SELECT regexp_like('ABC', '[A-Z]');
 regexp_like
-------------
 t
(1 row)
openGauss=# SELECT regexp_like('ABC', '[D-Z]');
 regexp_like
-------------
 f
(1 row)
openGauss=# SELECT regexp_like('ABC', '[a-z]','i');
 regexp_like
-------------
```

```
 t
(1 row)
```

- format(formatstr text [, str"any" [, ...]])

  Description: Formats a string.

  Return type: text

  Example:

```
openGauss=# SELECT format('Hello %s, %1$s', 'World');
      format
--------------------
 Hello World, World
(1 row)
```

- md5(string)

  Description: Encrypts a string in MD5 mode and returns a value in hexadecimal form.

  NOTE

  The MD5 encryption algorithm is not recommended because it has lower security and poses security risks.

  Return type: text

  Example:

```
openGauss=# SELECT md5('ABC');
             md5
----------------------------------
 902fbdd2b1df0c4f70b4a5d23525e932
(1 row)
```

- decode(string text, format text)

  Description: Decodes binary data from textual representation.

  Return type: bytea

  Example:

```
openGauss=# SELECT decode('MTIzAAE=', 'base64');
   decode
--------------
 \x3132330001
(1 row)
```

- similar_escape(pat text, esc text)

  Description: Converts a regular expression of the SQL:2008 style to the POSIX style.

  Return type: text

  Example:

```
openGauss=# select similar_escape('\s+ab','2');
 similar_escape
----------------
 ^(?:\\s+ab)$
(1 row)
```

- svals(hstore)

  Description: Obtains the value of the hstore type.

  Return type: SETOF text

  Example:

```
openGauss=# select svals('"aa"=>"bb"');
 svals
-------
```

```
 bb
(1 row)
```

- tconvert(key text, value text)

  Description: Converts character strings to the hstore format.

  Return type: hstore

  Example:

```
openGauss=# select tconvert('aa', 'bb');
  tconvert
------------
 "aa"=>"bb"
(1 row)
```

- encode(data bytea, format text)

  Description: Encodes binary data into a textual representation.

  Return type: text

  Example:

```
openGauss=# SELECT encode(E'123\\000\\001', 'base64');
  encode
----------
 MTIzAAE=
(1 row)
```

NOTE

- For a string containing newline characters, for example, a string consisting of a newline character and a space, the value of **length** and **lengthb** in GaussDB is 2.
- In GaussDB, *n* in the CHAR(n) type indicates the number of characters. Therefore, for multiple-octet coded character sets, the length returned by the LENGTHB function may be longer than *n*.
- GaussDB supports multiple types of databases, including A, B, C, and PG. If the database type is not specified, A is used by default. The lexical analyzer of A database is different from that of the other three databases. In A database, an empty character string is considered as **NULL**. Therefore, when a type A database is used, if a **NULL** character string is used as a parameter in the preceding character operation function, no output is displayed. For example:

```
openGauss=# SELECT translate('12345','123','');
translate
-----------

(1 row)
```

  This is because the kernel checks whether the input parameter contains **NULL** before calling the corresponding function. If yes, the kernel does not call the corresponding function. As a result, no output is displayed. In PG mode, the processing of character strings is the same as that of PostgreSQL. Therefore, the preceding problem does not occur.

11.5.4 Binary String Functions and Operators

String Operators

SQL defines some string functions that use keywords, rather than commas, to separate arguments.

- octet_length(string)

  Description: Specifies the number of bytes in a binary string.

  Return type: int

Example:

```
openGauss=# SELECT octet_length(E'jo\\000se'::bytea) AS RESULT;
 result
--------
      5
(1 row)
```

- overlay(string placing string from int [for int])

  Description: Replaces substrings.

  Return type: bytea

  Example:

```
openGauss=# SELECT overlay(E'Th\\000omas'::bytea placing E'\\002\\003'::bytea from 2 for 3) AS
RESULT;
     result
----------------
 \x5402036d6173
(1 row)
```

- position(substring in string)

  Description: Specifies the location of a specified substring.

  Return type: int

  Example:

```
openGauss=# SELECT position(E'\\000om'::bytea in E'Th\\000omas'::bytea) AS RESULT;
 result
--------
      3
(1 row)
```

- substring(string [from int] [for int])

  Description: Truncates a substring.

  Return type: bytea

  Example:

```
openGauss=# SELECT substring(E'Th\\000omas'::bytea from 2 for 3) AS RESULT;
  result
----------
 \x68006f
(1 row)
```

- substr(string, from int [, for int])

  Description: Truncates a substring.

  Return type: bytea

  Example:

```
openGauss=# select substr(E'Th\\000omas'::bytea,2, 3) as result;
  result
----------
 \x68006f
(1 row)
```

- trim([both] bytes from string)

  Description: Removes the longest string containing only bytes from **bytes** from the start and end of **string**.

  Return type: bytea

  Example:

```
openGauss=# SELECT trim(E'\\000'::bytea from E'\\000Tom\\000'::bytea) AS RESULT;
  result
----------
 \x546f6d
(1 row)
```

Other Binary String Functions

GaussDB provides common syntax used for calling functions.

- btrim(string bytea,bytes bytea)

  Description: Removes the longest string containing only bytes from **bytes** from the start and end of **string**.

  Return type: bytea

  Example:

```
openGauss=# SELECT btrim(E'\\000trim\\000'::bytea, E'\\000'::bytea) AS RESULT;
   result
------------
 \x7472696d
(1 row)
```

- get_bit(string, offset)

  Description: Extracts bits from a string.

  Return type: int

  Example:

```
openGauss=# SELECT get_bit(E'Th\\000omas'::bytea, 45) AS RESULT;
 result
--------
      1
(1 row)
```

- get_byte(string, offset)

  Description: Extracts bytes from a string.

  Return type: int

  Example:

```
openGauss=# SELECT get_byte(E'Th\\000omas'::bytea, 4) AS RESULT;
 result
--------
    109
(1 row)
```

- rawcmp

  Description: Specifies the raw data type comparison function.

  Parameter: raw, raw

  Return type: integer

- raweq

  Description: Specifies the raw data type comparison function.

  Parameter: raw, raw

  Return type: Boolean

- rawge

  Description: Specifies the raw data type comparison function.

  Parameter: raw, raw

  Return type: Boolean

- rawgt

  Description: Specifies the raw data type comparison function.

  Parameter: raw, raw

  Return type: Boolean

- rawin
  Description: Specifies the raw data type parsing function.
  Parameter: cstring
  Return type: bytea
- rawle
  Description: Specifies the raw data type parsing function.
  Parameter: raw, raw
  Return type: Boolean
- rawlike
  Description: Specifies the raw data type parsing function.
  Parameter: raw, raw
  Return type: Boolean
- rawlt
  Description: Specifies the raw data type parsing function.
  Parameter: raw, raw
  Return type: Boolean
- rawne
  Description: Compares whether the raw types are the same.
  Parameter: raw, raw
  Return type: Boolean
- rawnlike
  Description: Checks whether the raw type matches the mode.
  Parameter: raw, raw
  Return type: Boolean
- rawout
  Description: Specifies the RAW output API.
  Parameter: bytea
  Return type: cstring
- rawsend
  Description: Converts the bytea type to the binary type.
  Parameter: raw
  Return type: bytea
- rawtohex
  Description: Converts the raw format to the hexadecimal format.
  Parameter: text
  Return type: text
- set_bit(string,offset, newvalue)
  Description: Sets bits in a string.
  Return type: bytea
  Example:

```
openGauss=# SELECT set_bit(E'Th\\000omas'::bytea, 45, 0) AS RESULT;
      result
```

```
------------------
 \x5468006f6d4173
(1 row)
```

- set_byte(string,offset, newvalue)

  Description: Sets bytes in a string.

  Return type: bytea

  Example:

```
openGauss=# SELECT set_byte(E'Th\\000omas'::bytea, 4, 64) AS RESULT;
      result
------------------
 \x5468006f406173
(1 row)
```

11.5.5 Bit String Functions and Operators

Bit String Operators

Aside from the usual comparison operators, the following operators can be used. Bit string operands of **&**, **|**, and **#** must be of equal length. In case of bit shifting, the original length of the string is preserved by zero padding (if necessary).

- ||

  Description: Connects bit strings.

  Example:

```
openGauss=# SELECT B'10001' || B'011' AS RESULT;
  result
----------
 10001011
(1 row)
```

NOTE

It is recommended that a column have no more than 180 consecutive internal joins. A column with over 180 joins will be split into joined consecutive strings.

Example: **str1||str2||str3||str4** is split into **(str1||str2)||(str3||str4)**.

- &

  Description: Specifies the AND operation between bit strings.

  Example:

```
openGauss=# SELECT B'10001' & B'01101' AS RESULT;
 result
--------
 00001
(1 row)
```

- |

  Description: Specifies the OR operation between bit strings.

  Example:

```
openGauss=# SELECT B'10001' | B'01101' AS RESULT;
 result
--------
 11101
(1 row)
```

- #

Description: Specifies the OR operation between bit strings if they are inconsistent. If the same positions in the two bit strings are both 1 or 0, the position returns **0**.

Example:

```
openGauss=# SELECT B'10001' # B'01101' AS RESULT;
 result
--------
 11100
(1 row)
```

- ~

  Description: Specifies the NOT operation between bit strings.

  Example:

```
openGauss=# SELECT ~B'10001'AS RESULT;
 result
----------
 01110
(1 row)
```

- <<

  Description: Shifts left in a bit string.

  Example:

```
openGauss=# SELECT B'10001' << 3 AS RESULT;
 result
----------
 01000
(1 row)
```

- >>

  Description: Shifts right in a bit string.

  Example:

```
openGauss=# SELECT B'10001' >> 2 AS RESULT;
 result
----------
 00100
(1 row)
```

The following SQL-standard functions work on bit strings as well as strings: **length**, **bit_length**, **octet_length**, **position**, **substring**, and **overlay**.

The following functions work on bit strings as well as binary strings: **get_bit** and **set_bit**. When working with a bit string, these functions number the first (leftmost) bit of the string as bit 0.

In addition, it is possible to convert between integral values and type **bit**. Example:

```
openGauss=# SELECT 44::bit(10) AS RESULT;
   result
------------
 0000101100
(1 row)

openGauss=# SELECT 44::bit(3) AS RESULT;
 result
--------
 100
(1 row)

openGauss=# SELECT cast(-44 as bit(12)) AS RESULT;
    result
--------------
 111111010100
(1 row)
```

```
openGauss=# SELECT '1110'::bit(4)::integer AS RESULT;
 result
--------
     14
(1 row)

openGauss=# select substring('10101111'::bit(8), 2);
 substring
-----------
 0101111
(1 row)
```

NOTE

Casting to just "bit" means casting to bit(1), and so will deliver only the least significant bit of the integer.

11.5.6 Mode Matching Operators

There are three separate approaches to pattern matching provided by the database: the traditional SQL LIKE operator, the more recent SIMILAR TO operator, and POSIX-style regular expressions. Besides these basic operators, functions can be used to extract or replace matching substrings and to split a string at matching locations.

- LIKE

  Description: Specifies whether the string matches the mode string following **LIKE**. The LIKE expression returns true if the string matches the supplied pattern. (As expected, the NOT LIKE expression returns false if LIKE returns true, and vice versa.)

  Matching rules:

  a. This operator can succeed only when its pattern matches the entire string. If you want to match a sequence in any position within the string, the pattern must begin and end with a percent sign.
  b. The underscore (_) represents (matching) any single character. Percentage (%) indicates the wildcard character of any string.
  c. To match a literal underscore or percent sign without matching other characters, the respective character in **pattern** must be preceded by the escape character. The default escape character is the backslash but a different one can be selected by using the **ESCAPE** clause.
  d. To match with escape characters, enter two escape characters. For example, to write a **pattern** constant containing a backslash (\), you need to enter two backslashes in SQL statements.

NOTE

When **standard_conforming_strings** is set to **off**, any backslashes you write in literal string constants will need to be doubled. So, writing a pattern that matches a single backslash actually involves writing four backslashes in the statement (you can avoid this by selecting a different escape character with **ESCAPE** so that the backslash is no longer a special character of **LIKE**. But the backslash is still the special character of the character text analyzer, so you still need two backslashes.)

In MySQL-compatible schema, it is also possible to select no escape character by writing **ESCAPE ''**. This effectively disables the escape mechanism, which makes it impossible to turn off the special meaning of underscore and percent signs in the schema.

e. The keyword ILIKE can be used instead of LIKE to make the match case-insensitive.

f. Operator ~~ is equivalent to LIKE, and operator ~~* corresponds to ILIKE.

Example:

```
openGauss=# SELECT 'abc' LIKE 'abc' AS RESULT;
 result
-----------
 t
(1 row)
openGauss=# SELECT 'abc' LIKE 'a%' AS RESULT;
 result
-----------
 t
(1 row)
openGauss=# SELECT 'abc' LIKE '_b_' AS RESULT;
 result
-----------
 t
(1 row)
openGauss=# SELECT 'abc' LIKE 'c' AS RESULT;
 result
-----------
 f
(1 row)
```

- SIMILAR TO

  Description: Returns **true** or **false** depending on whether the pattern matches the given string. It is similar to **LIKE**, but differs in that **SIMILAR TO** uses the regular expression understanding pattern defined by the SQL standard.

  Matching rules:

  a. Similar to LIKE, this operator succeeds only when its pattern matches the entire string. If you want to match a sequence in any position within the string, the pattern must begin and end with a percent sign.

  b. The underscore (_) represents (matching) any single character. Percentage (%) indicates the wildcard character of any string.

  c. SIMILAR TO supports these pattern-matching metacharacters borrowed from POSIX regular expressions:

Metacharacter	Description
|	Specifies alternation (either of two alternatives).

Metacharacter	Description
*	Specifies repetition of the previous item zero or more times.
+	Specifies repetition of the previous item one or more times.
?	Specifies repetition of the previous item zero or one time.
{m}	Specifies repetition of the previous item exactly *m* times.
{m,}	Specifies repetition of the previous item *m* or more times.
{m,n}	Specifies repetition of the previous item at least *m* times and does not exceed *n* times.
()	Specifies that parentheses () can be used to group items into a single logical item.
[...]	Specifies a character class, just as in POSIX regular expressions.

d. A preamble escape character disables the special meaning of any of these metacharacters. The rules for using escape characters are the same as those for LIKE.

Regular expressions:

The **substring(string from pa...** function extracts a substring that matches an SQL regular expression pattern.

Example:

```
openGauss=# SELECT 'abc' SIMILAR TO 'abc' AS RESULT;
 result
-----------
 t
(1 row)
openGauss=# SELECT 'abc' SIMILAR TO 'a' AS RESULT;
 result
-----------
 f
(1 row)
openGauss=# SELECT 'abc' SIMILAR TO '%(b|d)%' AS RESULT;
 result
-----------
 t
(1 row)
openGauss=# SELECT 'abc' SIMILAR TO '(b|c)%'  AS RESULT;
 result
-----------
 f
(1 row)
```

- POSIX regular expressions

Description: A regular expression is a character sequence that is an abbreviated definition of a set of strings (a regular set). If a string is a member of a regular expression described by a regular expression, the string matches the regular expression. POSIX regular expressions provide a more powerful means for pattern matching than the LIKE and SIMILAR TO operators. **Table 11-29** lists all available operators for pattern matching using POSIX regular expressions.

Table 11-29 Regular expression match operators

Operator	Description	Example
~	Matches a regular expression, which is case-sensitive.	'thomas' ~ '.*thomas.*'
~*	Matches a regular expression, which is case-insensitive.	'thomas' ~* '.*Thomas.*'
!~	Does not match a regular expression, which is case-sensitive.	'thomas' !~ '.*Thomas.*'
!~*	Does not match a regular expression, which is case-sensitive.	'thomas' !~* '.*vadim.*'

Matching rules:

a. Unlike LIKE patterns, a regular expression is allowed to match anywhere within a string, unless the regular expression is explicitly anchored to the beginning or end of the string.

b. Besides the metacharacters mentioned above, POSIX regular expressions also support the following pattern matching metacharacters:

Metacharacter	Description
^	Specifies the match starting with a string.
$	Specifies the match at the end of a string.
.	Matches any single character.

Regular expressions:

POSIX regular expressions support the following functions:

– The **substring(string from pa...** function provides a method for extracting a substring that matches the POSIX regular expression pattern.

– The **•regexp_count(string tex...** function counts the number of substrings that match the POSIX regular expression pattern.

- The **•regexp_instr(string tex...** function obtains the position of a substring that matches the POSIX regular expression pattern.
- The **•regexp_substr(string te...** function extracts a substring that matches the POSIX regular expression pattern.
- The **regexp_replace(string, p...** function replaces a substring that matches the POSIX regular expression pattern with the new text.
- The **regexp_matches(string te...** function returns a text array consisting of all captured substrings that match a POSIX regular expression pattern.
- The **regexp_split_to_table(st...** function splits a string using a POSIX regular expression pattern as a delimiter.
- The **regexp_split_to_array(st...** function behaves the same as **regexp_split_to_table**, except that it returns its result as an array of text.

NOTE

The regular expression split functions ignore zero-length matches, which occur at the beginning or end of a string or after the previous match. This is contrary to the strict definition of regular expression matching. The latter is implemented by regexp_matches, but the former is usually the most commonly used behavior in practice.

Example:

```
openGauss=#  SELECT 'abc' ~ 'Abc' AS RESULT;
result
--------
 f
(1 row)
openGauss=# SELECT 'abc' ~* 'Abc' AS RESULT;
 result
--------
 t
(1 row)
openGauss=# SELECT 'abc' !~ 'Abc' AS RESULT;
 result
--------
 t
(1 row)
openGauss=# SELECT 'abc'!~* 'Abc' AS RESULT;
 result
--------
 f
(1 row)
openGauss=# SELECT 'abc' ~ '^a' AS RESULT;
 result
--------
 t
(1 row)
openGauss=# SELECT 'abc' ~ '(b|d)'AS RESULT;
 result
--------
 t
(1 row)
openGauss=# SELECT 'abc' ~ '^(b|c)'AS RESULT;
 result
--------
 f
(1 row)
```

Although most regular expression searches can be executed quickly, regular expressions can still be artificially made up of memory that takes a long time and any amount of memory. It is not recommended that you accept the regular expression search mode from the non-security mode source. If you must do this, you are advised to add the statement timeout limit. The search with the SIMILAR TO mode has the same security risks as the SIMILAR TO provides many capabilities that are the same as those of the POSIX- style regular expression. The LIKE search is much simpler than the other two options. Therefore, it is more secure to accept the non-secure mode source search.

11.5.7 Mathematical Functions and Operators

Numeric Operators

- +

  Description: Addition

  Example:

```
openGauss=# SELECT 2+3 AS RESULT;
 result
--------
      5
(1 row)
```

- -

  Description: Subtraction

  Example:

```
openGauss=# SELECT 2-3 AS RESULT;
 result
--------
     -1
(1 row)
```

- *

  Description: Multiplication

  Example:

```
openGauss=# SELECT 2*3 AS RESULT;
 result
--------
      6
(1 row)
```

- /

  Description: Division (The result is not rounded.)

  Example:

```
openGauss=# SELECT 4/2 AS RESULT;
 result
--------
      2
(1 row)
openGauss=# SELECT 4/3 AS RESULT;
      result
------------------
 1.33333333333333
(1 row)
```

- +/-

  Description: Positive/Negative

  Example:

```
openGauss=# SELECT -2 AS RESULT;
 result
--------
```

```
     -2
(1 row)
```

- %

  Description: Model (to obtain the remainder)

  Example:

```
openGauss=# SELECT 5%4 AS RESULT;
 result
--------
      1
(1 row)
```

- @

  Description: Absolute value

  Example:

```
openGauss=# SELECT @ -5.0 AS RESULT;
 result
--------
    5.0
(1 row)
```

- ^

  Description: Power (exponent calculation)

  Example:

```
openGauss=# SELECT 2.0^3.0 AS RESULT;
       result
--------------------
 8.0000000000000000
(1 row)
```

- |/

  Description: Square root

  Example:

```
openGauss=# SELECT |/ 25.0 AS RESULT;
 result
--------
      5
(1 row)
```

- ||/

  Description: Cubic root

  Example:

```
openGauss=# SELECT ||/ 27.0 AS RESULT;
 result
--------
      3
(1 row)
```

- !

  Description: Factorial

  Example:

```
openGauss=# SELECT 5! AS RESULT;
 result
--------
    120
(1 row)
```

- !!

  Description: Factorial (prefix operator)

  Example:

```
openGauss=# SELECT !!5 AS RESULT;
 result
--------
    120
(1 row)
```

- &

  Description: Binary AND

  Example:

```
openGauss=# SELECT 91&15  AS RESULT;
 result
--------
     11
(1 row)
```

- |

  Description: Binary OR

  Example:

```
openGauss=# SELECT 32|3  AS RESULT;
 result
--------
     35
(1 row)
```

- #

  Description: Binary XOR

  Example:

```
openGauss=# SELECT 17#5  AS RESULT;
 result
--------
     20
(1 row)
```

- ~

  Description: Binary NOT

  Example:

```
openGauss=# SELECT ~1 AS RESULT;
 result
--------
     -2
(1 row)
```

- <<

  Description: Binary shift left

  Example:

```
openGauss=# SELECT 1<<4 AS RESULT;
 result
--------
     16
(1 row)
```

- >>

  Description: Binary shift right

  Example:

```
openGauss=# SELECT 8>>2 AS RESULT;
 result
--------
      2
(1 row)
```

Numeric Operation Functions

- abs(x)

  Description: Absolute value

  Return type: same as the input

  Example:

  ```
  openGauss=# SELECT abs(-17.4);
   abs
  ------
   17.4
  (1 row)
  ```

- acos(x)

  Description: Arc cosine

  Return type: double precision

  Example:

  ```
  openGauss=# SELECT acos(-1);
         acos
  ------------------
   3.14159265358979
  (1 row)
  ```

- asin(x)

  Description: Arc sine

  Return type: double precision

  Example:

  ```
  openGauss=# SELECT asin(0.5);
         asin
  ------------------
   .523598775598299
  (1 row)
  ```

- atan(x)

  Description: Arc tangent

  Return type: double precision

  Example:

  ```
  openGauss=# SELECT atan(1);
         atan
  ------------------
   .785398163397448
  (1 row)
  ```

- atan2(y, x)

  Description: Arc tangent of y/x

  Return type: double precision

  Example:

  ```
  openGauss=# SELECT atan2(2, 1);
        atan2
  ------------------
   1.10714871779409
  (1 row)
  ```

- bitand(integer, integer)

  Description: Performs an AND (&) operation on two integers.

  Return type: bigint

  Example:

```
openGauss=# SELECT bitand(127, 63);
 bitand
--------
     63
(1 row)
```

- cbrt(dp)

  Description: Cubic root

  Return type: double precision

  Example:

```
openGauss=# SELECT cbrt(27.0);
 cbrt
------
    3
(1 row)
```

- ceil(x)

  Description: Minimum integer greater than or equal to the parameter

  Return type: integer

  Example:

```
openGauss=# SELECT ceil(-42.8);
 ceil
------
  -42
(1 row)
```

- ceiling(dp or numeric)

  Description: Minimum integer (alias of ceil) greater than or equal to the parameter

  Return type: dp or numeric. If implicit type conversion is not considered, the return type is the same as the input type.

  Example:

```
openGauss=# SELECT ceiling(-95.3);
 ceiling
---------
     -95
(1 row)
```

- cos(x)

  Description: Cosine

  Return type: double precision

  Example:

```
openGauss=# SELECT cos(-3.1415927);
        cos
-------------------
 -.999999999999999
(1 row)
```

- cot(x)

  Description: Cotangent

  Return type: double precision

  Example:

```
openGauss=# SELECT cot(1);
       cot
------------------
 .642092615934331
(1 row)
```

- degrees(dp)

  Description: Converts radians to angles.

  Return type: double precision

  Example:

```
openGauss=# SELECT degrees(0.5);
    degrees
------------------
 28.6478897565412
(1 row)
```

- div(y numeric, x numeric)

  Description: Integer part of y/x

  Return type: numeric

  Example:

```
openGauss=# SELECT div(9,4);
 div
-----
   2
(1 row)
```

- exp(x)

  Description: Natural exponent

  Return type: dp or numeric. If implicit type conversion is not considered, the return type is the same as the input type.

  Example:

```
openGauss=# SELECT exp(1.0);
       exp
--------------------
 2.7182818284590452
(1 row)
```

- floor(x)

  Description: Maximum integer not larger than the parameter

  Return type: same as the input

  Example:

```
openGauss=# SELECT floor(-42.8);
 floor
-------
   -43
(1 row)
```

- int1(in)

  Description: Converts the input text parameter to a value of the int1 type and returns the value.

  Return type: int1

  Example:

```
openGauss=# select int1('123');
 int1
------
 123
(1 row)
openGauss=# select int1('a');
 int1
------
   0
(1 row)
```

- int2(in)

  Description: Converts the input parameter to a value of the int2 type and returns the value.

  The supported input parameter types include float4, float8, int16, numeric, and text.

  Return type: int2

  Example:

```
openGauss=# select int2('1234');
 int2
------
 1234
(1 row)
openGauss=# select int2(25.3);
 int2
------
   25
(1 row)
```

- int4(in)

  Description: Converts the input parameter to a value of the int4 type and returns the value.

  The supported input parameter types include bit, boolean, char, double precision, int16, numeric, real, smallint and text

  Return type: int4

  Example:

```
openGauss=# select int4('789');
 int4
------
 789
(1 row)
openGauss=# select int4(99.9);
 int4
------
   99
(1 row)
```

- float4(in)

  Description: Converts the input parameter to a value of the float4 type and returns the value. The supported input parameter types include bigint, duoble precision, int16, integer, numeric, smallint, and text.

  Return type: float4

  Example:

```
openGauss=# select float4('789');
 float4
--------
    789
(1 row)

openGauss=# select float4(99.9);
 float4
--------
   99.9
(1 row)
```

- float8(in)

  Description: Converts the input parameter to a value of the float8 type and returns the value. The supported input parameter types include bigint, int16, integer, numeric, real, smallint, and text.

Return type: float8

Example:

```
openGauss=# select float8('789');
 float8
--------
    789
(1 row)

openGauss=# select float8(99.9);
 float8
--------
   99.9
(1 row)
```

- int16(in)

  Description: Converts the input parameter to a value of the int16 type and returns the value. The supported input parameter types include bigint, boolean, double precision, integer, numeric, oid, real, smallint, and tinyint.

  Return type: int16

  Example:

```
openGauss=# select int16('789');
 int16
--------
    789
(1 row)

openGauss=# select int16(99.9);
 int16
--------
   99
(1 row)
```

- numeric(in)

  Description: Converts the input parameter to a value of the numeric type and returns the value. The supported input parameter types include bigint, boolean, double precision, int16, integer, money, real, and smallint.

  Return type: numeric

  Example:

```
openGauss=# select "numeric"('789');
 numeric
---------
     789
(1 row)

openGauss=# select "numeric"(99.9);
 numeric
---------
    99.9
(1 row)
```

- oid(in)

  Description: Converts the input parameter to a value of the oid type and returns the value. The supported input parameter types include bigint and int16.

  Return type: oid

- radians(dp)

  Description: Converts angles to radians.

  Return type: double precision

Example:

```
openGauss=# SELECT radians(45.0);
     radians
------------------
 .785398163397448
(1 row)
```

- random()

  Description: Random number between 0.0 and 1.0

  Return type: double precision

  Example:

```
openGauss=# SELECT random();
      random
------------------
 .824823560658842
(1 row)
```

- multiply(x double precision or text, y double precision or text)

  Description: Product of x and y.

  Return type: double precision

  Example:

```
openGauss=# SELECT multiply(9.0, '3.0');
     multiply
-------------------
               27
(1 row)
openGauss=# SELECT multiply('9.0', 3.0);
     multiply
-------------------
               27
(1 row)
```

- ln(x)

  Description: Natural logarithm

  Return type: dp or numeric. If implicit type conversion is not considered, the return type is the same as the input type.

  Example:

```
openGauss=# SELECT ln(2.0);
        ln
-------------------
 .6931471805599453
(1 row)
```

- log(x)

  Description: Logarithm with 10 as the base

  Return type: same as the input

  Example:

```
openGauss=# SELECT log(100.0);
        log
--------------------
 2.0000000000000000
(1 row)
```

- log(b numeric, x numeric)

  Description: Logarithm with b as the base

  Return type: numeric

  Example:

```
openGauss=# SELECT log(2.0, 64.0);
        log
--------------------
 6.0000000000000000
(1 row)
```

- mod(x,y)

  Description: Remainder of x/y (model) If x equals to 0, 0 is returned.

  Return type: same as the parameter type

  Example:

```
openGauss=# SELECT mod(9,4);
 mod
-----
   1
(1 row)
openGauss=# SELECT mod(9,0);
 mod
-----
   9
(1 row)
```

- pi()

  Description: π constant value

  Return type: double precision

  Example:

```
openGauss=# SELECT pi();
        pi
------------------
 3.14159265358979
(1 row)
```

- power(a double precision, b double precision)

  Description: b power of a

  Return type: double precision

  Example:

```
openGauss=# SELECT power(9.0, 3.0);
        power
----------------------
 729.0000000000000000
(1 row)
```

- round(x)

  Description: Integer closest to the input parameter

  Return type: same as the input

  Example:

```
openGauss=# SELECT round(42.4);
 round
-------
    42
(1 row)

openGauss=# SELECT round(42.6);
 round
-------
    43
(1 row)
```

- round(v numeric, s int)

  Description: **s** digits are kept after the decimal point.

  Return type: numeric

Example:

```
openGauss=# SELECT round(42.4382, 2);
 round
-------
 42.44
(1 row)
```

- setseed(dp)

  Description: Sets seed for the following random() invoking (between –1.0 and 1.0, inclusive).

  Return type: void

  Example:

```
openGauss=# SELECT setseed(0.54823);
 setseed
---------

(1 row)
```

- sign(x)

  Description: Returns symbols of this parameter.

  Return type: **–1** indicates minus. **0** indicates 0, and **1** indicates positive numbers.

  Example:

```
openGauss=# SELECT sign(-8.4);
 sign
------
   -1
(1 row)
```

- sin(x)

  Description: Sine

  Return type: double precision

  Example:

```
openGauss=# SELECT sin(1.57079);
       sin
------------------
 .999999999979986
(1 row)
```

- sqrt(x)

  Description: Square root

  Return type: dp or numeric. If implicit type conversion is not considered, the return type is the same as the input type.

  Example:

```
openGauss=# SELECT sqrt(2.0);
       sqrt
-------------------
 1.414213562373095
(1 row)
```

- tan(x)

  Description: Tangent

  Return type: double precision

  Example:

```
openGauss=# SELECT tan(20);
       tan
------------------
```

```
 2.23716094422474
(1 row)
```

- trunc(x)

  Description: Truncates (the integral part).

  Return type: same as the input

  Example:

```
openGauss=# SELECT trunc(42.8);
 trunc
-------
    42
(1 row)
```

- trunc(v numeric, s int)

  Description: Truncates a number with **s** digits after the decimal point.

  Return type: numeric

  Example:

```
openGauss=# SELECT trunc(42.4382, 2);
 trunc
-------
 42.43
(1 row)
```

- smgrne(a smgr, b smgr)

  Description: Compares two integers of the smgr type to check whether they are different.

  Return type: Boolean

- smgreq(a smgr, b smgr)

  Description: Compares two integers of the smgr type to check whether they are equivalent.

  Return type: Boolean

- int1abs

  Description: Returns the absolute value of data of the uint8 type.

  Parameter: tinyint

  Return type: tinyint

- int1and

  Description: Returns the bitwise AND result of two data records of the uint8 type.

  Parameter: tinyint, tinyint

  Return type: tinyint

- int1cmp

  Description: Returns the comparison result of two data records of the uint8 type. If the value of the first parameter is greater, **1** is returned. If the value of the second parameter is greater, **–1** is returned. If they are the same, **0** is returned.

  Parameter: tinyint, tinyint

  Return type: integer

- int1div

  Description: Returns the result of dividing two data records of the uint8 type. The result is of the float8 type.

Parameter: tinyint, tinyint

Return type: tinyint

- int1eq

  Description: Compares two pieces of data of the uint8 type to check whether they are the same.

  Parameter: tinyint, tinyint

  Return type: Boolean

- int1ge

  Description: Determines whether the value of the first parameter is greater than or equal to the value of the second parameter in two data records of the uint8 type.

  Parameter: tinyint, tinyint

  Return type: Boolean

- int1gt

  Description: Performs a greater-than operation on an unsigned 1-byte integer.

  Parameter: tinyint, tinyint

  Return type: Boolean

- int1larger

  Description: Returns the maximum value of an unsigned 1-byte integer.

  Parameter: tinyint, tinyint

  Return type: tinyint

- int1le

  Description: Performs a less-than or an equal-to operation on an unsigned 1-byte integer.

  Parameter: tinyint, tinyint

  Return type: Boolean

- int1lt

  Description: Performs a less-than operation on an unsigned 1-byte integer.

  Parameter: tinyint, tinyint

  Return type: Boolean

- int1smaller

  Description: Calculates the minimum value of an unsigned 1-byte integer.

  Parameter: tinyint, tinyint

  Return type: tinyint

- int1inc

  Description: Unsigned 1-byte integer plus 1.

  Parameter: tinyint

  Return type: tinyint

- int1mi

  Description: Performs a minus operation on an unsigned 1-byte integer.

  Parameter: tinyint, tinyint

  Return type: tinyint

- int1mod

  Description: Performs a reminder operation on an unsigned 1-byte integer.

  Parameter: tinyint, tinyint

  Return type: tinyint
- int1mul

  Description: Performs a multiplication operation on unsigned 1-byte integer.

  Parameter: tinyint, tinyint

  Return type: tinyint
- int1ne

  Description: Performs a not-equal-to operation on an unsigned 1-byte integer.

  Parameter: tinyint, tinyint

  Return type: Boolean
- int1pl

  Description: Performs an addition operation on an unsigned 1-byte integer.

  Parameter: tinyint, tinyint

  Return type: tinyint
- int1um

  Description: Returns an unsigned 2-byte integer after subtracting the opposite number from the unsigned 1-byte integer.

  Parameter: tinyint

  Return type: smallint
- int1xor

  Description: Performs an exclusive OR operation on an unsigned 1-byte integer.

  Parameter: tinyint, tinyint

  Return type: tinyint
- cash_div_int1

  Description: Performs a division operation on the money type.

  Parameter: money, tinyint

  Return type: money
- cash_mul_int1

  Description: Performs a multiplication operation on the money type.

  Parameter: money, tinyint

  Return type: money
- int1not

  Description: Reverts binary bits of an unsigned 1-byte integer.

  Parameter: tinyint

  Return type: tinyint
- int1or

  Description: Performs an OR operation on an unsigned 1-byte integer.

  Parameter: tinyint, tinyint

Return type: tinyint

- int1shl

  Description: Shifts an unsigned 1-byte integer leftwards by a specified number of bits.

  Parameter: tinyint, integer

  Return type: tinyint

- int1shr

  Description: Shifts an unsigned 1-byte integer rightwards by a specified number of bits.

  Parameter: tinyint, integer

  Return type: tinyint

- width_bucket(op numeric, b1 numeric, b2 numeric, count int)

  Description: Returns a bucket to which the operand will be assigned in an equi-depth histogram with **count** buckets, ranging from **b1** to **b2**.

  Return type: int

  Example:

```
openGauss=# SELECT width_bucket(5.35, 0.024, 10.06, 5);
 width_bucket
--------------
            3
(1 row)
```

- width_bucket(op dp, b1 dp, b2 dp, count int)

  Description: Returns a bucket to which the operand will be assigned in an equi-depth histogram with **count** buckets, ranging from **b1** to **b2**.

  Return type: int

  Example:

```
openGauss=# SELECT width_bucket(5.35, 0.024, 10.06, 5);
 width_bucket
--------------
            3
(1 row)
```

11.5.8 Date and Time Processing Functions and Operators

Date and Time Operators

WARNING

When the user uses date/time operators, explicit type prefixes are modified for corresponding operands to ensure that the operands parsed by the database are consistent with what the user expects, and no unexpected results occur.

For example, abnormal mistakes will occur in the following example without an explicit data type.

```
SELECT date '2001-10-01' - '7' AS RESULT;
```

Table 11-30 Time and date operators

Operator	Example
+	openGauss=# SELECT date '2001-9-28' + integer '7' AS RESULT; result ------------ 2001-10-05 (1 row)
	openGauss=# SELECT date '2001-09-28' + interval '1 hour' AS RESULT; result --------------------- 2001-09-28 01:00:00 (1 row)
	openGauss=# SELECT date '2001-09-28' + time '03:00' AS RESULT; result --------------------- 2001-09-28 03:00:00 (1 row)
	openGauss=# SELECT interval '1 day' + interval '1 hour' AS RESULT; result ---------------- 1 day 01:00:00 (1 row)
	openGauss=# SELECT timestamp '2001-09-28 01:00' + interval '23 hours' AS RESULT; result --------------------- 2001-09-29 00:00:00 (1 row)
	openGauss=# SELECT time '01:00' + interval '3 hours' AS RESULT; result ---------- 04:00:00 (1 row)
-	openGauss=# SELECT date '2001-10-01' - date '2001-09-28' AS RESULT; result -------- 3days (1 row)
	openGauss=# SELECT date '2001-10-01' - integer '7' AS RESULT; result --------------------- 2001-09-24 00:00:00 (1 row)
	openGauss=# SELECT date '2001-09-28' - interval '1 hour' AS RESULT; result --------------------- 2001-09-27 23:00:00 (1 row)
	openGauss=# SELECT time '05:00' - time '03:00' AS RESULT; result ---------- 02:00:00 (1 row)

Operator	Example

	`openGauss=# SELECT time '05:00' - interval '2 hours' AS RESULT;`
	`  result`
	`----------`
	` 03:00:00`
	`(1 row)`

	`openGauss=# SELECT timestamp '2001-09-28 23:00' - interval '23 hours' AS RESULT;`
	`       result`
	`---------------------`
	` 2001-09-28 00:00:00`
	`(1 row)`

	`openGauss=# SELECT interval '1 day' - interval '1 hour' AS RESULT;`
	`  result`
	`----------`
	` 23:00:00`
	`(1 row)`

	`openGauss=# SELECT timestamp '2001-09-29 03:00' - timestamp '2001-09-27 12:00' AS RESULT;`
	`     result`
	`----------------`
	` 1 day 15:00:00`
	`(1 row)`

*	`openGauss=# SELECT 900 * interval '1 second' AS RESULT;`
	`  result`
	`----------`
	` 00:15:00`
	`(1 row)`

	`openGauss=# SELECT 21 * interval '1 day' AS RESULT;`
	` result`
	`---------`
	` 21 days`
	`(1 row)`

	`openGauss=# SELECT double precision '3.5' * interval '1 hour' AS RESULT;`
	`  result`
	`----------`
	` 03:30:00`
	`(1 row)`

/	`openGauss=# SELECT interval '1 hour' / double precision '1.5' AS RESULT;`
	`  result`
	`----------`
	` 00:40:00`
	`(1 row)`

Time/Date Functions

- age(timestamp, timestamp)

  Description: Subtracts parameters, producing a result in YYYY-MM-DD format. If the result is negative, the returned result is also negative. The input parameters can contain timezone or not.

  Return type: interval

  Example:

```
openGauss=# SELECT age(timestamp '2001-04-10', timestamp '1957-06-13');
           age
-------------------------
```

```
 43 years 9 mons 27 days
(1 row)
```

- age(timestamp)

  Description: Minuses the current time with the parameter. The input parameter can contain timezone or not.

  Return type: interval

  Example:

```
openGauss=# SELECT age(timestamp '1957-06-13');
          age
-------------------------
 60 years 2 mons 18 days
(1 row)
```

- clock_timestamp()

  Description: Specifies the current timestamp of the real-time clock.

  Return type: timestamp with time zone

  Example:

```
openGauss=# SELECT clock_timestamp();
       clock_timestamp
-------------------------------
 2017-09-01 16:57:36.636205+08
(1 row)
```

- current_date

  Description: Specifies the current date.

  Return type: date

  Example:

```
openGauss=# SELECT current_date;
    date
------------
 2017-09-01
(1 row)
```

- current_time

  Description: Specifies the current time.

  Return type: time with time zone

  Example:

```
openGauss=# SELECT current_time;
       timetz
--------------------
 16:58:07.086215+08
(1 row)
```

- current_timestamp

  Description: Specifies the current date and time.

  Return type: timestamp with time zone

  Example:

```
openGauss=# SELECT current_timestamp;
      pg_systimestamp
------------------------------
 2017-09-01 16:58:19.22173+08
(1 row)
```

- date_part(text, timestamp)

  Description: Obtains the value of a subdomain in date or time, for example, the year or hour. It is equivalent to **extract(field from timestamp)**.

Timestamp types: abstime, date, interval, reltime, time with time zone, time without time zone, timestamp with time zone, timestamp without time zone

Return type: double precision

Example:

```
openGauss=# SELECT date_part('hour', timestamp '2001-02-16 20:38:40');
 date_part
-----------
        20
(1 row)
```

- date_part(text, interval)

  Description: Obtains the subdomain value of the date/time value. When obtaining the month value, if the value is greater than 12, obtain the remainder after it is divided by 12. It is equivalent to **extract(field from timestamp)**.

  Return type: double precision

  Example:

```
openGauss=# SELECT date_part('month', interval '2 years 3 months');
 date_part
-----------
         3
(1 row)
```

- date_trunc(text, timestamp)

  Description: Truncates to the precision specified by **text**.

  Return type: interval, timestamp with time zone, timestamp without time zone

  Example:

```
openGauss=# SELECT date_trunc('hour', timestamp  '2001-02-16 20:38:40');
     date_trunc
---------------------
 2001-02-16 20:00:00
(1 row)
```

- trunc(timestamp)

  Description: Truncates to day by default.

  Example:

```
openGauss=# SELECT trunc(timestamp  '2001-02-16
20:38:40');
                         trunc
---------------------
2001-02-16 00:00:00
(1 row)
```

- trunc(arg1, arg2)

  Description: Truncates to the precision specified by **arg2**.

  Type of **arg1**: interval, timestamp with time zone, timestamp without time zone

  Type of **arg2**: text

  Return type: interval, timestamp with time zone, timestamp without time zone

  Example:

```
openGauss=# SELECT trunc(timestamp  '2001-02-16 20:38:40',
'hour');
                    trunc
---------------------
```

```
2001-02-16 20:00:00
(1 row)
```

- daterange(arg1, arg2)

  Description: Obtains time boundary information. The type of **arg1** and **arg2** is **date**.

  Return type: daterange

  Example:

```
openGauss=# select daterange('2000-05-06','2000-08-08');
       daterange
-------------------------
 [2000-05-06,2000-08-08)
(1 row)
```

- daterange(arg1, arg2, text)

  Description: Obtains time boundary information. The type of **arg1** and **arg2** is **date**, and the type of **text** is **text**.

  Return type: daterange

  Example:

```
openGauss=# select daterange('2000-05-06','2000-08-08','[]');
       daterange
-------------------------
 [2000-05-06,2000-08-09)
(1 row)
```

- extract(field from timestamp)

  Description: Obtains the hour.

  Return type: double precision

  Example:

```
openGauss=# SELECT extract(hour from timestamp '2001-02-16 20:38:40');
 date_part
-----------
        20
(1 row)
```

- extract(field from interval)

  Description: Obtains the month. If the value is greater than 12, obtain the remainder after it is divided by 12.

  Return type: double precision

  Example:

```
openGauss=# SELECT extract(month from interval '2 years 3 months');
 date_part
-----------
         3
(1 row)
```

- isfinite(date)

  Description: Tests for a valid date.

  Return type: Boolean

  Example:

```
openGauss=# SELECT isfinite(date '2001-02-16');
 isfinite
----------
 t
(1 row)
```

- isfinite(timestamp)

Description: Tests for a valid timestamp.

Return type: Boolean

Example:

```
openGauss=# SELECT isfinite(timestamp '2001-02-16 21:28:30');
 isfinite
----------
 t
(1 row)
```

- isfinite(interval)

  Description: Tests for a valid interval.

  Return type: Boolean

  Example:

```
openGauss=# SELECT isfinite(interval '4 hours');
 isfinite
----------
 t
(1 row)
```

- justify_days(interval)

  Description: Adjusts intervals to 30-day time periods, which are represented as months.

  Return type: interval

  Example:

```
openGauss=# SELECT justify_days(interval '35 days');
 justify_days
--------------
 1 mon 5 days
(1 row)
```

- justify_hours(interval)

  Description: Sets the time interval in days (24 hours is one day).

  Return type: interval

  Example:

```
openGauss=# SELECT JUSTIFY_HOURS(INTERVAL '27 HOURS');
 justify_hours
----------------
 1 day 03:00:00
(1 row)
```

- justify_interval(interval)

  Description: Adjusts **interval** using **justify_days** and **justify_hours**.

  Return type: interval

  Example:

```
openGauss=# SELECT JUSTIFY_INTERVAL(INTERVAL '1 MON -1 HOUR');
 justify_interval
------------------
 29 days 23:00:00
(1 row)
```

- localtime

  Description: Specifies the current time.

  Return type: time

  Example:

```
openGauss=# SELECT localtime AS RESULT;
     result
```

```
----------------
 16:05:55.664681
(1 row)
```

- localtimestamp

  Description: Specifies the current date and time.

  Return type: timestamp

  Example:

```
openGauss=# SELECT localtimestamp;
         timestamp
----------------------------
 2017-09-01 17:03:30.781902
(1 row)
```

- now()

  Description: Specifies the current date and time.

  Return type: timestamp with time zone

  Example:

```
openGauss=# SELECT now();
              now
-------------------------------
 2017-09-01 17:03:42.549426+08
(1 row)
```

- timenow

  Description: Specifies the current date and time.

  Return type: timestamp with time zone

  Example:

```
openGauss=# select timenow();
        timenow
------------------------
 2020-06-23 20:36:56+08
(1 row)
```

- numtodsinterval(num, interval_unit)

  Description: Converts a number to the interval type. **num** is a numeric-typed number. **interval_unit** is a string in the following format: 'DAY' | 'HOUR' | 'MINUTE' | 'SECOND'

  You can set the **IntervalStyle** parameter to **a** to be compatible with the interval output format of the function.

  Example:

```
openGauss=# SELECT numtodsinterval(100, 'HOUR');
 numtodsinterval
-----------------
 100:00:00
(1 row)

openGauss=# SET intervalstyle = a;
SET
openGauss=# SELECT numtodsinterval(100, 'HOUR');
        numtodsinterval
-------------------------------
 +000000004 04:00:00.000000000
(1 row)
```

- pg_sleep(seconds)

  Description: Specifies the delay time of the server thread in unit of second.

  Return type: void

Example:

```
openGauss=# SELECT pg_sleep(10);
 pg_sleep
----------

(1 row)
```

- statement_timestamp()

  Description: Specifies the current date and time.

  Return type: timestamp with time zone

  Example:

  ```
  openGauss=# SELECT statement_timestamp();
        statement_timestamp
  -------------------------------
   2017-09-01 17:04:39.119267+08
  (1 row)
  ```

- sysdate

  Description: Specifies the current date and time.

  Return type: timestamp

  Example:

  ```
  openGauss=# SELECT sysdate;
         sysdate
  ---------------------
   2017-09-01 17:04:49
  (1 row)
  ```

- timeofday()

  Description: Specifies the current date and time (like **clock_timestamp**, but returned as a **text** string)

  Return type: text

  Example:

  ```
  openGauss=# SELECT timeofday();
                timeofday
  -------------------------------------
   Fri Sep 01 17:05:01.167506 2017 CST
  (1 row)
  ```

- transaction_timestamp()

  Description: Specifies the current date and time (equivalent to **current_timestamp**)

  Return type: timestamp with time zone

  Example:

  ```
  openGauss=# SELECT transaction_timestamp();
       transaction_timestamp
  -------------------------------
   2017-09-01 17:05:13.534454+08
  (1 row)
  ```

- add_months(d,n)

  Description: Returns the date *date* plus *integer* months.

  d: indicates the value of the timestamp type and the value that can be implicitly converted to the timestamp type.

  n: indicates the value of the INTEGER type and the value that can be implicitly converted to the INTEGER type.

  Return type: timestamp

Example:

```
openGauss=# SELECT add_months(to_date('2017-5-29', 'yyyy-mm-dd'), 11) FROM sys_dummy;
     add_months
---------------------
 2018-04-29 00:00:00
(1 row)
```

- last_day(d)

  Description: Returns the date of the last day of the month that contains *date*.

  Return type: timestamp

  Example:

```
openGauss=# select last_day(to_date('2017-01-01', 'YYYY-MM-DD')) AS cal_result;
     cal_result
---------------------
 2017-01-31 00:00:00
(1 row)
```

- next_day(x,y)

  Description: Calculates the time of the next week y started from x.

  Return type: timestamp

  Example:

```
openGauss=# select next_day(timestamp '2017-05-25 00:00:00','Sunday')AS cal_result;
     cal_result
---------------------
 2017-05-28 00:00:00
(1 row)
```

- tinterval(abstime, abstime)

  Description: Creates a time interval with two pieces of absolute time.

  Return type: tinterval

  Example:

```
openGauss=# call tinterval(abstime 'May 10, 1947 23:59:12', abstime 'Mon May  1 00:30:30 1995');
                      tinterval
-----------------------------------------------------
 ["1947-05-10 23:59:12+09" "1995-05-01 00:30:30+08"]
(1 row)
```

- tintervalend(tinterval)

  Description: Returns the end time of **tinterval**.

  Return type: abstime

  Example:

```
openGauss=# select tintervalend('["Sep 4, 1983 23:59:12" "Oct4, 1983 23:59:12"]');
      tintervalend
------------------------
 1983-10-04 23:59:12+08
(1 row)
```

- tintervalrel(tinterval)

  Description: Calculates and returns the relative time of **tinterval**.

  Return type: reltime

  Example:

```
openGauss=# select tintervalrel('["Sep 4, 1983 23:59:12" "Oct4, 1983 23:59:12"]');
 tintervalrel
--------------
 1 mon
(1 row)
```

- smalldatetime_ge

  Description: Checks whether the value of the first parameter is greater than or equal to that of the second parameter.

  Parameter: smalldatetime, smalldatetime

  Return type: Boolean

- smalldatetime_cmp

  Description: Compares two smalldatetime values to check whether they are the same.

  Parameter: smalldatetime, smalldatetime

  Return type: integer

- smalldatetime_eq

  Description: Compares two smalldatetime values to check whether they are the same.

  Parameter: smalldatetime, smalldatetime

  Return type: Boolean

- smalldatetime_gt

  Description: Determines whether the first parameter is greater than the second.

  Parameter: smalldatetime, smalldatetime

  Return type: Boolean

- smalldatetime_hash

  Description: Calculates the hash value corresponding to a timestamp.

  Parameter: smalldatetime

  Return type: integer

- smalldatetime_in

  Description: Inputs a timestamp.

  Parameter: cstring, oid, integer

  Return type: smalldatetime

- smalldatetime_larger

  Description: Returns a larger timestamp.

  Parameter: smalldatetime, smalldatetime

  Return type: smalldatetime

- smalldatetime_le

  Description: Checks whether the value of the first parameter is less than or equal to that of the second parameter.

  Parameter: smalldatetime, smalldatetime

  Return type: Boolean

- smalldatetime_lt

  Description: Determines whether the first parameter is less than the second parameter.

  Parameter: smalldatetime, smalldatetime

Return type: Boolean

- smalldatetime_ne

  Description: Compares two timestamps to check whether they are different.

  Parameter: smalldatetime, smalldatetime

  Return type: Boolean

- smalldatetime_out

  Description: Converts a timestamp into the external form.

  Parameter: smalldatetime

  Return type: cstring

- smalldatetime_send

  Description: Converts a timestamp to the binary format.

  Parameter: smalldatetime

  Return type: bytea

- smalldatetime_smaller

  Description: Returns a smaller smalldatetime.

  Parameter: smalldatetime, smalldatetime

  Return type: smalldatetime

- smalldatetime_to_abstime

  Description: Converts smalldatetime to abstime.

  Parameter: smalldatetime

  Return type: abstime

- smalldatetime_to_time

  Description: Converts smalldatetime to time.

  Parameter: smalldatetime

  Return type: time without time zone

- smalldatetime_to_timestamp

  Description: Converts smalldatetime to timestamp.

  Parameter: smalldatetime

  Return type: timestamp without time zone

- smalldatetime_to_timestamptz

  Description: Converts smalldatetime to timestamptz.

  Parameter: smalldatetime

  Return type: timestamp with time zone

- smalldatetime_to_varchar2

  Description: Converts smalldatetime to varchar2.

  Parameter: smalldatetime

  Return type: character varying

NOTE

There are multiple methods for obtaining the current time. Select an appropriate API based on the actual service scenario.

1. The following APIs return values based on the start time of the current transaction:
   ```
   CURRENT_DATE CURRENT_TIME CURRENT_TIME(precision)
   CURRENT_TIMESTAMP(precision) LOCALTIME LOCALTIMESTAMP LOCALTIME(precision)
   LOCALTIMESTAMP(precision)
   ```
   The values transferred by CURRENT_TIME and CURRENT_TIMESTAMP(precision) contain time zone information. The values transferred by LOCALTIME and LOCALTIMESTAMP do not contain time zone information. CURRENT_TIME, LOCALTIME, and LOCALTIMESTAMP can be optionally attached with a precision parameter, which rounds the second field of the result to the specified decimal place. If there is no precision parameter, the result is given the full precision that can be obtained.

   Because these functions all return results by the start time of the current transaction, their values do not change throughout the transaction. We think this is a feature with the purpose to allow a transaction to have a consistent concept at the "current" time, so that multiple modifications in the same transaction can maintain the same timestamp.
2. The following APIs return the start time of the current statement:
   ```
   transaction_timestamp() statement_timestamp() now()
   ```
   transaction_timestamp() is equivalent to CURRENT_TIMESTAMP(precision), and its name clearly reflects its return value. statement_timestamp() returns the start time of the current statement (more accurately, the time when the last instruction is received from the client). The return values of **statement_timestamp()** and **transaction_timestamp()** are the same during the execution of the first instruction of a transaction, but may be different in subsequent instructions.

   now() is equivalent to **transaction_timestamp()**.

   1. The following APIs return the actual current time when the function is invoked:
      ```
      clock_timestamp() timeofday()
      ```
      clock_timestamp() returns the actual current time, and its value changes even in the same SQL statement. Similar to **clock_timestamp()**, **timeofday()** also returns the actual current time. However, the result of **timeofday()** is a formatted text string instead of a timestamp with time zone information.

Table 11-31 shows the templates for truncating date/time values.

Table 11-31 Truncating date/time values

Item	Format	Description
Microsecond	MICROSECON	Truncates date/time values, accurate to the microsecond (000000–999999).
	US	
	USEC	
	USECOND	
Millisecond	MILLISECON	Truncates date/time values, accurate to the millisecond (000–999).
	MS	
	MSEC	
	MSECOND	

Item	Format	Description
Second	S	Truncates date/time values, accurate to the second (00–59).
	SEC	
	SECOND	
Minute	M	Truncates date/time values, accurate to the minute (00–59).
	MI	
	MIN	
	MINUTE	
Hour	H	Truncates date/time values, accurate to the hour (00–23).
	HH	
	HOUR	
	HR	
Day	D	Truncates date/time values, accurate to the day (01-01 to 12–31)
	DAY	
	DD	
	DDD	
	J	
Week	W	Truncates date/time values, accurate to the week (the first day of the current week).
	WEEK	
Month	MM	Truncates date/time values, accurate to the month (the first day of the current month).
	MON	
	MONTH	
Quarter	Q	Truncates date/time values, accurate to the quarter (the first day of the current quarter).
	QTR	
	QUARTER	
Year	Y	Truncates date/time values, accurate to the year (the first day of the current year).
	YEAR	
	YR	
	YYYY	
Decade	DEC	Truncates date/time values, accurate to the decade (the first day of the current decade).
	DECADE	

Item	Format	Description
Century	C	Truncates date/time values, accurate to the century (the first day of the current century).
	CC	
	CENT	
	CENTURY	
Millennium	MIL	Truncates date/time values, accurate to the millennium (the first day of the current millennium).
	MILLENNIA	
	MILLENNIUM	

TIMESTAMPDIFF

- **TIMESTAMPDIFF(***unit , timestamp_expr1, timestamp_expr2***)**

The **timestampdiff** function returns the result of **timestamp_expr2 – timestamp_expr1** in the specified unit. **timestamp_expr1** and **timestamp_expr2** must be value expressions of the **timestamp**, **timestamptz**, or **date** type. **unit** determines the unit of the difference between two dates.

NOTE

This function is valid only when GaussDB is compatible with the MY type (that is, dbcompatibility = 'B').

- year

  Year.

```
openGauss=#  SELECT TIMESTAMPDIFF(YEAR, '2018-01-01', '2020-01-01');
 timestamp_diff
----------------
              2
(1 row)
```

- quarter

  Quarter.

```
openGauss=#  SELECT TIMESTAMPDIFF(QUARTER, '2018-01-01', '2020-01-01');
 timestamp_diff
----------------
              8
(1 row)
```

- month

  Month.

```
openGauss=#  SELECT TIMESTAMPDIFF(MONTH, '2018-01-01', '2020-01-01');
 timestamp_diff
----------------
             24
(1 row)
```

- week

  Week.

```
openGauss=#  SELECT TIMESTAMPDIFF(WEEK, '2018-01-01', '2020-01-01');
 timestamp_diff
```

```
----------------
          104
(1 row)
```

- day

  Day.

```
openGauss=#  SELECT TIMESTAMPDIFF(DAY, '2018-01-01', '2020-01-01');
 timestamp_diff
----------------
          730
(1 row)
```

- hour

  Hour.

```
openGauss=#  SELECT TIMESTAMPDIFF(HOUR, '2020-01-01 10:10:10', '2020-01-01 11:11:11');
 timestamp_diff
----------------
            1
(1 row)
```

- minute

  Minute.

```
openGauss=#  SELECT TIMESTAMPDIFF(MINUTE, '2020-01-01 10:10:10', '2020-01-01 11:11:11');
 timestamp_diff
----------------
           61
(1 row)
```

- second

  Second.

```
openGauss=#  SELECT TIMESTAMPDIFF(SECOND, '2020-01-01 10:10:10', '2020-01-01 11:11:11');
 timestamp_diff
----------------
         3661
(1 row)
```

- microseconds

  The seconds column, including fractional parts, is multiplied by 1,000,000.

```
openGauss=#  SELECT TIMESTAMPDIFF(MICROSECOND, '2020-01-01 10:10:10.000000', '2020-01-01 
10:10:10.111111');
 timestamp_diff
----------------
       111111
(1 row)
```

- timestamp_expr with the time zone

```
openGauss=# SELECT TIMESTAMPDIFF(HOUR,'2020-05-01 10:10:10-01','2020-05-01 10:10:10-03');
 timestamp_diff
----------------
            2
(1 row)
```

EXTRACT

- **EXTRACT(***field* **FROM** *source***)**

The **extract** function retrieves subcolumns such as year or hour from date/time values. **source** must be a value expression of type **timestamp**, **time**, or **interval**. (Expressions of type **date** are cast to **timestamp** and can therefore be used as well.) **field** is an identifier or string that selects what column to extract from the source value. The **extract** function returns values of type **double precision**. The following are valid **field** names:

- century

  The first century starts at 0001-01-01 00:00:00 AD. This definition applies to all Gregorian calendar countries. There is no century number 0. You go from **–1** century to **1** century.

  Example:

```
openGauss=# SELECT EXTRACT(CENTURY FROM TIMESTAMP '2000-12-16 12:21:13');
 date_part
-----------
        20
(1 row)
```

- day
  - For **timestamp** values, the day (of the month) column (1–31)

```
openGauss=# SELECT EXTRACT(DAY FROM TIMESTAMP '2001-02-16 20:38:40');
 date_part
-----------
        16
(1 row)
```

  - For **interval** values, the number of days

```
openGauss=# SELECT EXTRACT(DAY FROM INTERVAL '40 days 1 minute');
 date_part
-----------
        40
(1 row)
```

- decade

  Year column divided by 10

```
openGauss=# SELECT EXTRACT(DECADE FROM TIMESTAMP '2001-02-16 20:38:40');
 date_part
-----------
       200
(1 row)
```

- dow

  Day of the week as Sunday (**0**) to Saturday (**6**)

```
openGauss=# SELECT EXTRACT(DOW FROM TIMESTAMP '2001-02-16 20:38:40');
 date_part
-----------
         5
(1 row)
```

- doy

  Day of the year (1–365 or 366)

```
openGauss=# SELECT EXTRACT(DOY FROM TIMESTAMP '2001-02-16 20:38:40');
 date_part
-----------
        47
(1 row)
```

- epoch
  - For **timestamp with time zone** values, the number of seconds since 1970-01-01 00:00:00-00 UTC (can be negative).

    For **date** and **timestamp** values, the number of seconds since 1970-01-01 00:00:00-00 local time.

    For **interval** values, the total number of seconds in the interval.

```
openGauss=# SELECT EXTRACT(EPOCH FROM TIMESTAMP WITH TIME ZONE '2001-02-16
20:38:40.12-08');
  date_part
--------------
 982384720.12
(1 row)
```

```
openGauss=# SELECT EXTRACT(EPOCH FROM INTERVAL '5 days 3 hours');
 date_part
-----------
    442800
(1 row)
```

– Way to convert an epoch value back to a timestamp

```
openGauss=# SELECT TIMESTAMP WITH TIME ZONE 'epoch' + 982384720.12 * INTERVAL '1
second' AS RESULT;
          result
---------------------------
 2001-02-17 12:38:40.12+08
(1 row)
```

- hour

  Hour column (0–23)

```
openGauss=# SELECT EXTRACT(HOUR FROM TIMESTAMP '2001-02-16 20:38:40');
 date_part
-----------
        20
(1 row)
```

- isodow

  Day of the week (1–7)

  Monday is 1 and Sunday is 7.

  NOTE

  This is identical to **dow** except for Sunday.

```
openGauss=# SELECT EXTRACT(ISODOW FROM TIMESTAMP '2001-02-18 20:38:40');
 date_part
-----------
         7
(1 row)
```

- isoyear

  The ISO 8601 year that the date falls in (not applicable to intervals).

  Each ISO year begins with the Monday of the week containing January 4, so in early January or late December the ISO year may be different from the Gregorian year. See the **week** column for more information.

```
openGauss=# SELECT EXTRACT(ISOYEAR FROM DATE '2006-01-01');
 date_part
-----------
      2005
(1 row)
openGauss=# SELECT EXTRACT(ISOYEAR FROM DATE '2006-01-02');
 date_part
-----------
      2006
(1 row)
```

- microseconds

  The seconds column, including fractional parts, is multiplied by 1,000,000.

```
openGauss=# SELECT EXTRACT(MICROSECONDS FROM TIME '17:12:28.5');
 date_part
-----------
  28500000
(1 row)
```

- millennium

  Years in the 1900s are in the second millennium. The third millennium started from 0:00 of January 1, 2001.

```
openGauss=# SELECT EXTRACT(MILLENNIUM FROM TIMESTAMP '2001-02-16 20:38:40');
 date_part
```

```
-----------
        3
(1 row)
```

- milliseconds

  Seconds column, including fractional parts, is multiplied by 1000. Note that this includes full seconds.

```
openGauss=# SELECT EXTRACT(MILLISECONDS FROM TIME '17:12:28.5');
 date_part
-----------
     28500
(1 row)
```

- minute

  Minutes column (0–59).

```
openGauss=# SELECT EXTRACT(MINUTE FROM TIMESTAMP '2001-02-16 20:38:40');
 date_part
-----------
       38
(1 row)
```

- month

  For **timestamp** values, the specific month in the year (1–12).

```
openGauss=# SELECT EXTRACT(MONTH FROM TIMESTAMP '2001-02-16 20:38:40');
 date_part
-----------
        2
(1 row)
```

  For **interval** values, the number of months, modulo 12 (0–11).

```
openGauss=# SELECT EXTRACT(MONTH FROM INTERVAL '2 years 13 months');
 date_part
-----------
        1
(1 row)
```

- quarter

  Quarter of the year (1–4) that the date is in.

```
openGauss=# SELECT EXTRACT(QUARTER FROM TIMESTAMP '2001-02-16 20:38:40');
 date_part
-----------
        1
(1 row)
```

- second

  Seconds column, including fractional parts (0–59).

```
openGauss=# SELECT EXTRACT(SECOND FROM TIME '17:12:28.5');
 date_part
-----------
      28.5
(1 row)
```

- timezone

  Time zone offset from UTC, measured in seconds. Positive values correspond to time zones east of UTC, negative values to zones west of UTC.

- timezone_hour

  Hour component of the time zone offset.

- timezone_minute

  Minute component of the time zone offset.

- week

Number of the week of the year that the day is in. By definition (ISO 8601), the first week of a year contains January 4 of that year. (The ISO-8601 week starts on Monday.) In other words, the first Thursday of a year is in week 1 of that year.

Because of this, it is possible for early January dates to be part of the 52nd or 53rd week of the previous year, and late December dates to be part of the 1st week of the next year. For example, **2005-01-01** is part of the 53rd week of year 2004, **2006-01-01** is part of the 52nd week of year 2005, and **2012-12-31** is part of the 1st week of year 2013. You are advised to use the columns **isoyear** and **week** together to ensure consistency.

```
openGauss=# SELECT EXTRACT(WEEK FROM TIMESTAMP '2001-02-16 20:38:40');
 date_part
-----------
         7
(1 row)
```

- year

  Year column.

```
openGauss=# SELECT EXTRACT(YEAR FROM TIMESTAMP '2001-02-16 20:38:40');
 date_part
-----------
      2001
(1 row)
```

date_part

The **date_part** function is modeled on the traditional Ingres equivalent to the SQL-standard function **extract**:

- **date_part('*field*', *source*)**

Note that here the **field** parameter needs to be a string value, not a name. The valid field names for **date_part** are the same as for **extract**. For details, see **EXTRACT**.

Example:

```
openGauss=# SELECT date_part('day', TIMESTAMP '2001-02-16 20:38:40');
 date_part
-----------
        16
(1 row)
openGauss=# SELECT date_part('hour', INTERVAL '4 hours 3 minutes');
 date_part
-----------
         4
(1 row)
```

Table 11-32 specifies the schema for formatting date and time values.

Table 11-32 Schema for formatting date and time

Category	Format	Description
Hour	HH	Number of hours in one day (01–12)
	HH12	Number of hours in one day (01–12)
	HH24	Number of hours in one day (00–23)

Category	Format	Description
Minute	MI	Minute (00–59)
Second	SS	Second (00–59)
	FF	Microsecond (000000–999999)
	SSSSS	Second after midnight (0–86399)
Morning and afternoon	AM or A.M.	Morning identifier
	PM or P.M.	Afternoon identifier
Year	Y,YYY	Year with comma (with four digits or more)
	SYYYY	Year with four digits BC
	YYYY	Year (with four digits or more)
	YYY	Last three digits of a year
	YY	Last two digits of a year
	Y	Last one digit of a year
	IYYY	ISO year (with four digits or more)
	IYY	Last three digits of an ISO year
	IY	Last two digits of an ISO year
	I	Last one digit of an ISO year
	RR	Last two digits of a year (A year of the 20th century can be stored in the 21st century.)
	RRRR	Capable of receiving a year with four digits or two digits. If there are 2 digits, the value is the same as the returned value of RR. If there are 4 digits, the value is the same as YYYY.
	● BC or B.C. ● AD or A.D.	Era indicator Before Christ (BC) and After Christ (AD)
Month	MONTH	Full spelling of a month in uppercase (9 characters are filled in if the value is empty.)
	MON	Month in abbreviated format in uppercase (with three characters)
	MM	Month (01–12)
	RM	Month in Roman numerals (I–XII; I=JAN) and uppercase
Day	DAY	Full spelling of a date in uppercase (9 characters are filled in if the value is empty.)

Category	Format	Description
	DY	Day in abbreviated format in uppercase (with three characters)
	DDD	Day in a year (001–366)
	DD	Day in a month (01–31)
	D	Day in a week (1–7).
Week	W	Week in a month (1–5) (The first week starts from the first day of the month.)
	WW	Week in a year (1–53) (The first week starts from the first day of the year.)
	IW	Week in an ISO year (The first Thursday is in the first week.)
Century	CC	Century (with two digits) (The 21st century starts from 2001-01-01.)
Julian date	J	Julian date (starting from January 1 of 4712 BC)
Quarter	Q	Quarter

NOTE

In the table, the rules for RR to calculate years are as follows:

- If the range of the input two-digit year is between 00 and 49:

  If the last two digits of the current year are between 00 and 49, the first two digits of the returned year are the same as the first two digits of the current year.

  If the last two digits of the current year are between 50 and 99, the first two digits of the returned year equal to the first two digits of the current year plus 1.
- If the range of the input two-digit year is between 50 and 99:

  If the last two digits of the current year are between 00 and 49, the first two digits of the returned year equal to the first two digits of the current year minus 1.

  If the last two digits of the current year are between 50 and 99, the first two digits of the returned year are the same as the first two digits of the current year.

11.5.9 Type Conversion Functions

Type Conversion Functions

- cash_words(money)

  Description: Type conversion function, which converts money into text.

  Example:

```
openGauss=# SELECT cash_words('1.23');
           cash_words
-----------------------------------
 One dollar and twenty three cents
(1 row)
```

- cast(x as y)

  Description: Converts x into the type specified by y.

  Example:

```
openGauss=# SELECT cast('22-oct-1997' as timestamp);
      timestamp
---------------------
 1997-10-22 00:00:00
(1 row)
```

- hextoraw(raw)

  Description: Converts a string in hexadecimal format into raw type.

  Return type: raw

  Example:

```
openGauss=# SELECT hextoraw('7D');
 hextoraw
----------
 7D
(1 row)
```

- numtoday(numeric)

  Description: Converts values of the number type into the timestamp of the specified type.

  Return type: timestamp

  Example:

```
openGauss=# SELECT numtoday(2);
 numtoday
----------
 2 days
(1 row)
```

- pg_systimestamp()

  Description: Obtains the system timestamp.

  Return type: timestamp with time zone

  Example:

```
openGauss=# SELECT pg_systimestamp();
        pg_systimestamp
-------------------------------
 2015-10-14 11:21:28.317367+08
(1 row)
```

- rawtohex(string)

  Description: Converts a string in binary format into hexadecimal format.

  The result is the ACSII code of the input characters in hexadecimal format.

  Return type: varchar

  Example:

```
openGauss=# SELECT rawtohex('1234567');
    rawtohex
----------------
 31323334353637
(1 row)
```

- to_bigint(varchar)

  Description: Converts the character type to the bigint type.

  Return type: bigint

  Example:

```
openGauss=# SELECT to_bigint('123364545554455');
    to_bigint
----------------
 123364545554455
(1 row)
```

- to_char(datetime/interval [, fmt])

  Description: Converts a DATETIME or INTERVAL value of the DATE/TIMESTAMP/TIMESTAMP WITH TIME ZONE/TIMESTAMP WITH LOCAL TIME ZONE type into the VARCHAR type according to the format specified by **fmt**.

  – The optional parameter **fmt** allows for the following types: date, time, week, quarter, and century. Each type has a unique template. The templates can be combined together. Common templates include HH, MI, SS, YYYY, MM, and DD.

  – A template may have a modification word. FM is a common modification word and is used to suppress the preceding zero or the following blank spaces.

  Return type: varchar

  Example:

```
openGauss=# SELECT to_char(current_timestamp,'HH12:MI:SS');
 to_char
----------
 10:19:26
(1 row)
openGauss=# SELECT to_char(current_timestamp,'FMHH12:FMMI:FMSS');
 to_char
----------
 10:19:46
(1 row)
```

- to_char(double precision/real, text)

  Description: Converts the values of the floating point type into the strings in the specified format.

  Return type: text

  Example:

```
openGauss=# SELECT to_char(125.8::real, '999D99');
 to_char
---------
  125.80
(1 row)
```

- to_char(numeric/smallint/integer/bigint/double precision/real[, fmt])

  Descriptions: Converts an integer or a value in floating point format into a string in specified format.

  – The optional parameter **fmt** allows for the following types: decimal characters, grouping characters, positive/negative sign and currency sign. Each type has a unique template. The templates can be combined together. Common templates include: 9, 0, millesimal sign (,), and decimal point (.).

  – A template can have a modification word, similar to FM. However, FM does not suppress 0 which is output according to the template.

  – Use the template X or x to convert an integer value into a string in hexadecimal format.

  Return type: varchar

  Example:

```
openGauss=# SELECT to_char(1485,'9,999');
 to_char
---------
  1,485
(1 row)
openGauss=# SELECT to_char( 1148.5,'9,999.999');
  to_char
------------
  1,148.500
(1 row)
openGauss=# SELECT to_char(148.5,'990999.909');
   to_char
-------------
    0148.500
(1 row)
openGauss=# SELECT to_char(123,'XXX');
 to_char
---------
   7B
(1 row)
```

- to_char(interval, text)

  Description: Converts the values of the time interval type into the strings in the specified format.

  Return type: text

  Example:

```
openGauss=# SELECT to_char(interval '15h 2m 12s', 'HH24:MI:SS');
 to_char
----------
 15:02:12
(1 row)
```

- to_char(int, text)

  Description: Converts the values of the integer type into the strings in the specified format.

  Return type: text

  Example:

```
openGauss=# SELECT to_char(125, '999');
 to_char
---------
  125
(1 row)
```

- to_char(numeric, text)

  Description: Converts the values of the numeric type into the strings in the specified format.

  Return type: text

  Example:

```
openGauss=# SELECT to_char(-125.8, '999D99S');
 to_char
---------
 125.80-
(1 row)
```

- to_char(string)

  Description: Converts the CHAR/VARCHAR/VARCHAR2/CLOB type into the VARCHAR type.

  If this function is used to convert data of the CLOB type, and the value to be converted exceeds the value range of the target type, an error is returned.

  Return type: varchar

Example:

```
openGauss=# SELECT to_char('01110');
 to_char
---------
 01110
(1 row)
```

- to_char(timestamp, text)

  Description: Converts the values of the timestamp type into the strings in the specified format.

  Return type: text

  Example:

```
openGauss=# SELECT to_char(current_timestamp, 'HH12:MI:SS');
 to_char
----------
 10:55:59
(1 row)
```

- to_clob(char/nchar/varchar/varchar2/nvarchar/nvarchar2/text/raw)

  Description: Converts the raw type or text character set type CHAR, NCHAR, VARCHAR, VARCHAR2, NVARCHAR, NVARCHAR2, or TEXT to the CLOB type.

  Return type: clob

  Example:

```
openGauss=# SELECT to_clob('ABCDEF'::RAW(10));
 to_clob
---------
 ABCDEF
(1 row)
openGauss=# SELECT to_clob('hello111'::CHAR(15));
 to_clob
----------
 hello111
(1 row)
openGauss=# SELECT to_clob('gauss123'::NCHAR(10));
 to_clob
----------
 gauss123
(1 row)
openGauss=# SELECT to_clob('gauss234'::VARCHAR(10));
 to_clob
----------
 gauss234
(1 row)
openGauss=# SELECT to_clob('gauss345'::VARCHAR2(10));
 to_clob
----------
 gauss345
(1 row)
openGauss=# SELECT to_clob('gauss456'::NVARCHAR2(10));
 to_clob
----------
 gauss456
(1 row)
openGauss=# SELECT to_clob('World222!'::TEXT);
  to_clob
-----------
 World222!
(1 row)
```

- to_date(text)

  Description: Converts values of the text type into the timestamp in the specified format. Currently, only the following two formats are supported:

  - Format 1: Date without separators, for example, 20150814. The value must contain the complete year, month, and day.
  - Format 2: Date with separators, for example, 2014-08-14. The separator can be any non-digit character.

  Return type: timestamp without time zone

  Example:

```
openGauss=# SELECT to_date('2015-08-14');
       to_date
---------------------
 2015-08-14 00:00:00
(1 row)
```

- to_date(text, text)

  Description: Converts the values of the string type into the dates in the specified format.

  Return type: timestamp without time zone

  Example:

```
openGauss=# SELECT to_date('05 Dec 2000', 'DD Mon YYYY');
       to_date
---------------------
 2000-12-05 00:00:00
(1 row)
```

- to_number (expr [, fmt])

  Description: Converts **expr** into a value of the NUMBER type according to the specified format.

  For details about the type conversion formats, see **Table 11-33**.

  If a hexadecimal string is converted into a decimal number, the hexadecimal string can include a maximum of 16 bytes if it is to be converted into a sign-free number.

  During the conversion from a hexadecimal string to a decimal digit, the format string cannot have a character other than x or X. Otherwise, an error is reported.

  Return type: number

  Example:

```
openGauss=# SELECT to_number('12,454.8-', '99G999D9S');
 to_number
-----------
  -12454.8
(1 row)
```

- to_number(text, text)

  Description: Converts the values of the string type into the numbers in the specified format.

  Return type: numeric

  Example:

```
openGauss=# SELECT to_number('12,454.8-', '99G999D9S');
 to_number
-----------
  -12454.8
(1 row)
```

- to_timestamp(double precision)

  Description: Converts a UNIX century into a timestamp.

Return type: timestamp with time zone

Example:

```
openGauss=# SELECT to_timestamp(1284352323);
      to_timestamp
------------------------
 2010-09-13 12:32:03+08
(1 row)
```

- to_timestamp(string [,fmt])

  Description: Converts a string into a value of the timestamp type according to the format specified by **fmt**. When **fmt** is not specified, perform the conversion according to the format specified by **nls_timestamp_format**.

  In **to_timestamp** in GaussDB,

  – If the input year *YYYY* is 0, an error will be reported.

  – If the input year *YYYY* is less than 0, specify *SYYYY* in **fmt**. The year with the value of n (an absolute value) BC will be output correctly.

  Characters in the **fmt** must match the schema for formatting the data and time. Otherwise, an error is reported.

  Return type: timestamp without time zone

  Example:

```
openGauss=# SHOW nls_timestamp_format;
    nls_timestamp_format
----------------------------
 DD-Mon-YYYY HH:MI:SS.FF AM
(1 row)

openGauss=# SELECT to_timestamp('12-sep-2014');
    to_timestamp
---------------------
 2014-09-12 00:00:00
(1 row)
openGauss=# SELECT to_timestamp('12-Sep-10 14:10:10.123000','DD-Mon-YY HH24:MI:SS.FF');
      to_timestamp
-------------------------
 2010-09-12 14:10:10.123
(1 row)
openGauss=# SELECT to_timestamp('-1','SYYYY');
      to_timestamp
------------------------
 0001-01-01 00:00:00 BC
(1 row)
openGauss=# SELECT to_timestamp('98','RR');
    to_timestamp
---------------------
 1998-01-01 00:00:00
(1 row)
openGauss=# SELECT to_timestamp('01','RR');
    to_timestamp
---------------------
 2001-01-01 00:00:00
(1 row)
```

- to_timestamp(text, text)

  Description: Converts values of the string type into the timestamp of the specified type.

  Return type: timestamp

  Example:

```
openGauss=# SELECT to_timestamp('05 Dec 2000', 'DD Mon YYYY');
    to_timestamp
```

```
---------------------
 2000-12-05 00:00:00
(1 row)
```

Table 11-33 Template patterns for numeric formatting

Pattern	Description
9	Value with specified digits
0	Values with leading zeros
Period (.)	Decimal point
Comma (,)	Group (thousand) separator
PR	Negative values in angle brackets
S	Sign anchored to number (uses locale)
L	Currency symbol (uses locale)
D	Decimal point (uses locale)
G	Group separator (uses locale)
MI	Minus sign in the specified position (if the number is less than 0)
PL	Plus sign in the specified position (if the number is greater than 0)
SG	Plus or minus sign in the specified position
RN	Roman numerals (the input values are between 1 and 3999)
TH or th	Ordinal number suffix
V	Shifts specified number of digits (decimal)

- abstime_text

  Description: Converts abstime to text.

  Parameter: abstime

  Return type: text
- abstime_to_smalldatetime

  Description: Converts abstime to smalldatatime.

  Parameter: abstime

  Return type: smalldatetime
- bigint_tid

  Description: Converts bigint to tid.

  Parameter: bigint

  Return type: tid

- bool_int1
  Description: Converts bool to int1.
  Parameter: Boolean
  Return type: tinyint
- bool_int2
  Description: Converts bool to int2.
  Parameter: Boolean
  Return type: smallint
- bool_int8
  Description: Converts bool to int8.
  Parameter: Boolean
  Return type: bigint
- bpchar_date
  Description: Converts a string to a date.
  Parameter: character
  Return type: date
- bpchar_float4
  Description: Converts a string to float4.
  Parameter: character
  Return type: real
- bpchar_float8
  Description: Converts a string to float8.
  Parameter: character
  Return type: double precision
- bpchar_int4
  Description: Converts a string to int4.
  Parameter: character
  Return type: integer
- bpchar_int8
  Description: Converts a string to int8.
  Parameter: character
  Return type: bigint
- bpchar_numeric
  Description: Converts a string to numeric.
  Parameter: character
  Return type: numeric
- bpchar_timestamp
  Description: Converts a string to a timestamp.
  Parameter: character
  Return type: timestamp without time zone

- bpchar_to_smalldatetime

  Description: Converts a string to smalldatetime.

  Parameter: character

  Return type: smalldatetime

- complex_array_in

  Description: Converts the external complex_array type to the internal anyarray array type.

  Parameter: cstring, oid, int2vector

  Return type: anyarray

- cupointer_bigint

  Description: Converts the column-store CU pointer type to the bigint type.

  Parameter: text

  Return type: bigint

- date_bpchar

  Description: Converts the date type to bpchar.

  Parameter: date

  Return type: character

- date_text

  Description: Converts date to text.

  Parameter: date

  Return type: text

- date_varchar

  Description: Converts date to varchar.

  Parameter: date

  Return type: character varying

- f4toi1

  Description: Forcibly converts float4 to uint8.

  Parameter: real

  Return type: tinyint

- f8toi1

  Description: Forcibly converts float8 to uint8.

  Parameter: double precision

  Return type: tinyint

- float4_bpchar

  Description: Converts float4 to bpchar.

  Parameter: real

  Return type: character

- float4_text

  Description: Converts float4 to text.

  Parameter: real

  Return type: text

- float4_varchar
  Description: Converts float4 to varchar.
  Parameter: real
  Return type: character varying
- float8_bpchar
  Description: Converts float8 to bpchar.
  Parameter: double precision
  Return type: character
- float8_interval
  Description: Converts float8 to interval.
  Parameter: double precision
  Return type: interval
- float8_text
  Description: Converts float8 to text.
  Parameter: double precision
  Return type: text
- float8_varchar
  Description: Converts float8 to varchar.
  Parameter: double precision
  Return type: character varying
- i1tof4
  Description: Converts uint8 to float4.
  Parameter: tinyint
  Return type: real
- i1tof8
  Description: Converts uint8 to float8.
  Parameter: tinyint
  Return type: double precision
- i1toi2
  Description: Converts uint8 to int16.
  Parameter: tinyint
  Return type: smallint
- i1toi4
  Description: Converts uint8 to int32.
  Parameter: tinyint
  Return type: integer
- i1toi8
  Description: Converts uint8 to int64.
  Parameter: tinyint
  Return type: bigint

- i2toi1

  Description: Converts int16 to uint8.

  Parameter: smallint

  Return type: tinyint

- i4toi1

  Description: Converts int32 to uint8.

  Parameter: integer

  Return type: tinyint

- i8toi1

  Description: Converts int64 to uint8.

  Parameter: bigint

  Return type: tinyint

- int1_avg_accum

  Description: Adds the second parameter of the uint8 type to the first parameter. The first parameter is an array of the bigint type.

  Parameter: bigint[], tinyint

  Return type: bigint[]

- int1_bool

  Description: Converts uint8 to bool.

  Parameter: tinyint

  Return type: Boolean

- int1_bpchar

  Description: Converts uint8 to bpchar.

  Parameter: tinyint

  Return type: character

- int1_mul_cash

  Description: Returns the product of a parameter of the int8 type and a parameter of the cash type. The return type is cash.

  Parameter: tinyint, money

  Return type: money

- int1_numeric

  Description: Converts uint8 to numeric.

  Parameter: tinyint

  Return type: numeric

- int1_nvarchar2

  Description: Converts uint8 to nvarchar2.

  Parameter: tinyint

  Return type: nvarchar2

- int1_text

  Description: Converts uint8 to text.

  Parameter: tinyint

Return type: text

- int1_varchar

  Description: Converts uint8 to varchar.

  Parameter: tinyint

  Return type: character varying
- int1in

  Description: Converts a string into an unsigned 1-byte integer.

  Parameter: cstring

  Return type: tinyint
- int1out

  Description: Converts an unsigned 1-byte integer into a string.

  Parameter: tinyint

  Return type: cstring
- int1up

  Description: Converts an input integer to an unsigned 1-byte integer.

  Parameter: tinyint

  Return type: tinyint
- int2_bool

  Description: Converts a signed two-byte integer to the bool type.

  Parameter: smallint

  Return type: Boolean
- int2_bpchar

  Description: Converts a signed two-byte integer to the bpchar type.

  Parameter: smallint

  Return type: character
- int2_text

  Description: Converts a signed two-byte integer to the text type.

  Parameter: smallint

  Return type: text
- int2_varchar

  Description: Converts a signed two-byte integer to the varchar type.

  Parameter: smallint

  Return type: character varying
- int4_bpchar

  Description: Converts a signed four-byte integer to bpchar.

  Parameter: integer

  Return type: character
- int4_text

  Description: Converts a signed four-byte integer to the text type.

  Parameter: integer

  Return type: text

- int4_varchar

  Description: Converts a signed four-byte integer into varchar.

  Parameter: integer

  Return type: character varying

- int8_bool

  Description: Converts an eight-byte signed integer to a Boolean value.

  Parameter: bigint

  Return type: Boolean

- int8_bpchar

  Description: Converts an 8-byte signed integer to bpchar.

  Parameter: bigint

  Return type: character

- int8_text

  Description: Converts an eight-byte signed integer to the text type.

  Parameter: bigint

  Return type: text

- int8_varchar

  Description: Converts an eight-byte signed integer to varchar.

  Parameter: bigint

  Return type: character varying

- intervaltonum

  Description: Converts the internal dats type date to numeric.

  Parameter: interval

  Return type: numeric

- numeric_bpchar

  Description: Converts numeric to bpchar.

  Parameter: numeric

  Return type: character

- numeric_int1

  Description: Converts numeric to a signed one-byte integer.

  Parameter: numeric

  Return type: tinyint

- numeric_text

  Description: Converts numeric to text.

  Parameter: numeric

  Return type: text

- numeric_varchar

  Description: Converts numeric to varchar.

  Parameter: numeric

  Return type: character varying

- nvarchar2in

  Description: Converts c string to varchar.

  Parameter: cstring, oid, integer

  Return type: nvarchar2

- nvarchar2out

  Description: Converts text into a c string.

  Parameter: nvarchar2

  Return type: cstring

- nvarchar2send

  Description: Converts varchar to binary.

  Parameter: nvarchar2

  Return type: bytea

- oidvectorin_extend

  Description: Converts a string to oidvector.

  Parameter: cstring

  Return type: oidvector_extend

- oidvectorout_extend

  Description: Converts oidvector to a string.

  Parameter: oidvector_extend

  Return type: cstring

- oidvectorsend_extend

  Description: Converts oidvector to a string.

  Parameter: oidvector_extend

  Return type: bytea

- reltime_text

  Description: Converts reltime to text.

  Parameter: reltime

  Return type: text

- text_date

  Description: Converts the text type to the date type.

  Parameter: text

  Return type: date

- text_float4

  Description: Converts text to float4.

  Parameter: text

  Return type: real

- text_float8

  Description: Converts the text type to float8.

  Parameter: text

  Return type: double precision

- text_int1
  Description: Converts the text type to int1.
  Parameter: text
  Return type: tinyint
- text_int2
  Description: Converts the text type to the int2 type.
  Parameter: text
  Return type: smallint
- text_int4
  Description: Converts the text type to int4.
  Parameter: text
  Return type: integer
- text_int8
  Description: Converts the text type to the int8 type.
  Parameter: text
  Return type: bigint
- text_numeric
  Description: Converts the text type to the numeric type.
  Parameter: text
  Return type: numeric
- text_timestamp
  Description: Converts the text type to the timestamp type.
  Parameter: text
  Return type: timestamp without time zone
- time_text
  Description: Converts the time type to the text type.
  Parameter: time without time zone
  Return type: text
- timestamp_text
  Description: Converts the timestamp type to the text type.
  Parameter: timestamp without time zone
  Return type: text
- timestamp_to_smalldatetime
  Description: Converts the timestamp type to the smalldatetime type.
  Parameter: timestamp without time zone
  Return type: smalldatetime
- timestamp_varchar
  Description: Converts the timestamp type to varchar.
  Parameter: timestamp without time zone
  Return type: character varying

- timestamptz_to_smalldatetime

  Description: Converts timestamptz to smalldatetime.

  Parameter: timestamp with time zone

  Return type: smalldatetime

- timestampzone_text

  Description: Converts the timestampzone type to the text type.

  Parameter: timestamp with time zone

  Return type: text

- timetz_text

  Description: Converts the timetz type to the text type.

  Parameter: time with time zone

  Return type: text

- to_integer

  Description: Converts data to the integer type.

  Parameter: character varying

  Return type: integer

- to_interval

  Description: Converts to the interval type.

  Parameter: character varying

  Return type: interval

- to_numeric

  Description: Converts to the numeric type.

  Parameter: character varying

  Return type: numeric

- to_nvarchar2

  Description: Converts to the nvarchar2 type.

  Parameter: numeric

  Return type: nvarchar2

- to_text

  Description: Converts to the text type.

  Parameter: smallint

  Return type: text

- to_ts

  Description: Converts to the ts type.

  Parameter: character varying

  Return type: timestamp without time zone

- to_varchar2

  Description: Converts to the varchar2 type.

  Parameter: timestamp without time zone

  Return type: character varying

- varchar_date

  Description: Converts varchar to date.

  Parameter: character varying

  Return type: date

- varchar_float4

  Description: Converts varchar to float4.

  Parameter: character varying

  Return type: real

- varchar_float8

  Description: Converts the varchar type to the float8 type.

  Parameter: character varying

  Return type: double precision

- varchar_int4

  Description: Converts the type from varchar to int4.

  Parameter: character varying

  Return type: integer

- varchar_int8

  Description: Converts the varchar type to the int8 type.

  Parameter: character varying

  Return type: bigint

- varchar_numeric

  Description: Converts varchar to numeric.

  Parameter: character varying

  Return type: numeric

- varchar_timestamp

  Description: Converts varchar to timestamp.

  Parameter: character varying

  Return type: timestamp without time zone

- varchar2_to_smlldatetime

  Description: Converts varchar2 to smlldatetime.

  Parameter: character varying

  Return type: smalldatetime

- xidout4

  Description: The xid output is a four-byte number.

  Parameter: xid32

  Return type: cstring

- xidsend4

  Description: Converts xid to the binary format.

  Parameter: xid32

  Return type: bytea

Encoding Type Conversion

- convert_to_nocase(text, text)

  Description: Converts a string into a specified encoding type.

  Return type: bytea

  Example:

  ```
  openGauss=# SELECT convert_to_nocase('12345', 'GBK');
   convert_to_nocase
  -------------------
   \x3132333435
  (1 row)
  ```

11.5.10 Geometric Functions and Operators

Geometric Operators

- +

  Description: Translation

  Example:

  ```
  openGauss=# SELECT box '((0,0),(1,1))' + point '(2.0,0)' AS RESULT;
     result
  -------------
   (3,1),(2,0)
  (1 row)
  ```

- -

  Description: Translation

  Example:

  ```
  openGauss=# SELECT box '((0,0),(1,1))' - point '(2.0,0)' AS RESULT;
      result
  ---------------
   (-1,1),(-2,0)
  (1 row)
  ```

- *

  Description: Scaling out/Rotation

  Example:

  ```
  openGauss=# SELECT box '((0,0),(1,1))' * point '(2.0,0)' AS RESULT;
     result
  -------------
   (2,2),(0,0)
  (1 row)
  ```

- /

  Description: Scaling in/Rotation

  Example:

  ```
  openGauss=# SELECT box '((0,0),(2,2))' / point '(2.0,0)' AS RESULT;
     result
  -------------
   (1,1),(0,0)
  (1 row)
  ```

- #

  Description: Intersection of two figures

  Example:

  ```
  openGauss=# SELECT box '((1,-1),(-1,1))' # box '((1,1),(-2,-2))' AS RESULT;
   result
  ```

```
---------------
 (1,1),(-1,-1)
(1 row)
```

- #

  Description: Number of paths or polygon vertexes of a figure

  Example:

```
openGauss=# SELECT # path'((1,0),(0,1),(-1,0))' AS RESULT;
 result
--------
      3
(1 row)
```

- @-@

  Description: Length or circumference of a figure

  Example:

```
openGauss=# SELECT @-@ path '((0,0),(1,0))' AS RESULT;
 result
--------
      2
(1 row)
```

- @@

  Description: Center of a figure

  Example:

```
openGauss=# SELECT @@ circle '((0,0),10)' AS RESULT;
 result
--------
 (0,0)
(1 row)
```

- <->

  Description: Distance between two figures

  Example:

```
openGauss=# SELECT circle '((0,0),1)' <-> circle '((5,0),1)' AS RESULT;
 result
--------
      3
(1 row)
```

- &&

  Description: Overlaps? (One point in common makes this true.)

  Example:

```
openGauss=# SELECT box '((0,0),(1,1))' && box '((0,0),(2,2))' AS RESULT;
 result
--------
 t
(1 row)
```

- <<

  Description: Is strictly left of (no common horizontal coordinate)?

  Example:

```
openGauss=# SELECT circle '((0,0),1)' << circle '((5,0),1)' AS RESULT;
 result
--------
 t
(1 row)
```

- >>

  Description: Is strictly right of (no common horizontal coordinate)?

Example:

```
openGauss=# SELECT circle '((5,0),1)' >> circle '((0,0),1)' AS RESULT;
 result
--------
 t
(1 row)
```

- &<

  Description: Does not extend to the right of?

  Example:

```
openGauss=# SELECT box '((0,0),(1,1))' &< box '((0,0),(2,2))' AS RESULT;
 result
--------
 t
(1 row)
```

- &>

  Description: Does not extend to the left of?

  Example:

```
openGauss=# SELECT box '((0,0),(3,3))' &> box '((0,0),(2,2))' AS RESULT;
 result
--------
 t
(1 row)
```

- <<|

  Description: Is strictly below (no common horizontal coordinate)?

  Example:

```
openGauss=# SELECT box '((0,0),(3,3))' <<| box '((3,4),(5,5))' AS RESULT;
 result
--------
 t
(1 row)
```

- |>>

  Description: Is strictly above (no common horizontal coordinate)?

  Example:

```
openGauss=# SELECT box '((3,4),(5,5))' |>> box '((0,0),(3,3))' AS RESULT;
 result
--------
 t
(1 row)
```

- &<|

  Description: Does not extend above?

  Example:

```
openGauss=# SELECT box '((0,0),(1,1))' &<| box '((0,0),(2,2))' AS RESULT;
 result
--------
 t
(1 row)
```

- |&>

  Description: Does not extend below?

  Example:

```
openGauss=# SELECT box '((0,0),(3,3))' |&> box '((0,0),(2,2))' AS RESULT;
 result
--------
 t
(1 row)
```

- <^

  Description: Is below (allows touching)?

  Example:

```
openGauss=# SELECT box '((0,0),(-3,-3))' <^ box '((0,0),(2,2))' AS RESULT;
 result
--------
 t
(1 row)
```

- >^

  Description: Is above (allows touching)?

  Example:

```
openGauss=# SELECT box '((0,0),(2,2))' >^ box '((0,0),(-3,-3))'  AS RESULT;
 result
--------
 t
(1 row)
```

- ?#

  Description: Intersect?

  Example:

```
openGauss=# SELECT lseg '((-1,0),(1,0))' ?# box '((-2,-2),(2,2))' AS RESULT;
 result
--------
 t
(1 row)
```

- ?-

  Description: Is horizontal?

  Example:

```
openGauss=# SELECT ?- lseg '((-1,0),(1,0))' AS RESULT;
 result
--------
 t
(1 row)
```

- ?-

  Description: Are horizontally aligned?

  Example:

```
openGauss=# SELECT point '(1,0)' ?- point '(0,0)' AS RESULT;
 result
--------
 t
(1 row)
```

- ?|

  Description: Is vertical?

  Example:

```
openGauss=# SELECT ?| lseg '((-1,0),(1,0))' AS RESULT;
 result
--------
 f
(1 row)
```

- ?|

  Description: Are vertically aligned?

  Example:

```
openGauss=# SELECT point '(0,1)' ?| point '(0,0)' AS RESULT;
 result
```

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

- ?-|

  Description: Are perpendicular?

  Example:

  ```
  openGauss=# SELECT lseg '((0,0),(0,1))' ?-| lseg '((0,0),(1,0))' AS RESULT;
   result
  --------
   t
  (1 row)
  ```

- ?||

  Description: Are parallel?

  Example:

  ```
  openGauss=# SELECT lseg '((-1,0),(1,0))' ?|| lseg '((-1,2),(1,2))' AS RESULT;
   result
  --------
   t
  (1 row)
  ```

- @>

  Description: Contains?

  Example:

  ```
  openGauss=# SELECT circle '((0,0),2)' @> point '(1,1)' AS RESULT;
   result
  --------
   t
  (1 row)
  ```

- <@

  Description: Contained in or on?

  Example:

  ```
  openGauss=# SELECT point '(1,1)' <@ circle '((0,0),2)' AS RESULT;
   result
  --------
   t
  (1 row)
  ```

- ~=

  Description: Same as?

  Example:

  ```
  openGauss=# SELECT polygon '((0,0),(1,1))' ~= polygon '((1,1),(0,0))' AS RESULT;
   result
  --------
   t
  (1 row)
  ```

Geometric Functions

- area(object)

  Description: Area calculation

  Return type: double precision

  Example:

  ```
  openGauss=# SELECT area(box '((0,0),(1,1))') AS RESULT;
   result
  --------
  ```

```
      1
(1 row)
```

- center(object)

  Description: Figure center calculation

  Return type: point

  Example:

  ```
  openGauss=# SELECT center(box '((0,0),(1,2))') AS RESULT;
   result
  ---------
   (0.5,1)
  (1 row)
  ```

- diameter(circle)

  Description: Circle diameter calculation

  Return type: double precision

  Example:

  ```
  openGauss=# SELECT diameter(circle '((0,0),2.0)') AS RESULT;
   result
  --------
        4
  (1 row)
  ```

- height(box)

  Description: Vertical size of box

  Return type: double precision

  Example:

  ```
  openGauss=# SELECT height(box '((0,0),(1,1))') AS RESULT;
   result
  --------
        1
  (1 row)
  ```

- isclosed(path)

  Description: A closed path?

  Return type: Boolean

  Example:

  ```
  openGauss=# SELECT isclosed(path '((0,0),(1,1),(2,0))') AS RESULT;
   result
  --------
   t
  (1 row)
  ```

- isopen(path)

  Description: An open path?

  Return type: Boolean

  Example:

  ```
  openGauss=# SELECT isopen(path '[(0,0),(1,1),(2,0)]') AS RESULT;
   result
  --------
   t
  (1 row)
  ```

- length(object)

  Description: Length calculation

  Return type: double precision

  Example:

```
openGauss=# SELECT length(path '((-1,0),(1,0))') AS RESULT;
 result
--------
      4
(1 row)
```

- npoints(path)

  Description: Number of points in a path

  Return type: int

  Example:

```
openGauss=# SELECT npoints(path '[(0,0),(1,1),(2,0)]') AS RESULT;
 result
--------
      3
(1 row)
```

- npoints(polygon)

  Description: Number of points in a polygon

  Return type: int

  Example:

```
openGauss=# SELECT npoints(polygon '((1,1),(0,0))') AS RESULT;
 result
--------
      2
(1 row)
```

- pclose(path)

  Description: Converts a path to closed.

  Return type: path

  Example:

```
openGauss=# SELECT pclose(path '[(0,0),(1,1),(2,0)]') AS RESULT;
       result
---------------------
 ((0,0),(1,1),(2,0))
(1 row)
```

- popen(path)

  Description: Converts a path to open.

  Return type: path

  Example:

```
openGauss=# SELECT popen(path '((0,0),(1,1),(2,0))') AS RESULT;
       result
---------------------
 [(0,0),(1,1),(2,0)]
(1 row)
```

- radius(circle)

  Description: Circle radius calculation

  Return type: double precision

  Example:

```
openGauss=# SELECT radius(circle '((0,0),2.0)') AS RESULT;
 result
--------
      2
(1 row)
```

- width(box)

  Description: Horizontal size of a box

Return type: double precision

Example:

```
openGauss=# SELECT width(box '((0,0),(1,1))') AS RESULT;
 result
--------
      1
(1 row)
```

Geometric Type Conversion Functions

- box(circle)

  Description: Circle to box

  Return type: box

  Example:

```
openGauss=# SELECT box(circle '((0,0),2.0)') AS RESULT;
                                   result
---------------------------------------------------------------------------
 (1.41421356237309,1.41421356237309),(-1.41421356237309,-1.41421356237309)
(1 row)
```

- box(point, point)

  Description: Points to box

  Return type: box

  Example:

```
openGauss=# SELECT box(point '(0,0)', point '(1,1)') AS RESULT;
   result
-------------
 (1,1),(0,0)
(1 row)
```

- box(polygon)

  Description: Polygon to box

  Return type: box

  Example:

```
openGauss=# SELECT box(polygon '((0,0),(1,1),(2,0))') AS RESULT;
   result
-------------
 (2,1),(0,0)
(1 row)
```

- circle(box)

  Description: Box to circle

  Return type: circle

  Example:

```
openGauss=# SELECT circle(box '((0,0),(1,1))') AS RESULT;
           result
-------------------------------
 <(0.5,0.5),0.707106781186548>
(1 row)
```

- circle(point, double precision)

  Description: Center and radius to circle

  Return type: circle

  Example:

```
openGauss=# SELECT circle(point '(0,0)', 2.0) AS RESULT;
  result
```

```
-----------
 <(0,0),2>
(1 row)
```

- circle(polygon)

  Description: Polygon to circle

  Return type: circle

  Example:

```
openGauss=# SELECT circle(polygon '((0,0),(1,1),(2,0))') AS RESULT;
                  result
-------------------------------------------
 <(1,0.333333333333333),0.924950591148529>
(1 row)
```

- lseg(box)

  Description: Box diagonal to line segment

  Return type: lseg

  Example:

```
openGauss=# SELECT lseg(box '((-1,0),(1,0))') AS RESULT;
     result
----------------
 [(1,0),(-1,0)]
(1 row)
```

- lseg(point, point)

  Description: Points to line segment

  Return type: lseg

  Example:

```
openGauss=# SELECT lseg(point '(-1,0)', point '(1,0)') AS RESULT;
     result
----------------
 [(-1,0),(1,0)]
(1 row)
```

- slope(point, point)

  Description: Calculates the slope of a straight line formed by two points.

  Return type: double

  Example:

```
openGauss=# SELECT slope(point '(1,1)', point '(0,0)') AS RESULT;
 result
--------
      1
(1 row)
```

- path(polygon)

  Description: Polygon to path

  Return type: path

  Example:

```
openGauss=# SELECT path(polygon '((0,0),(1,1),(2,0))') AS RESULT;
       result
---------------------
 ((0,0),(1,1),(2,0))
(1 row)
```

- point(double precision, double precision)

  Description: Points

  Return type: point

Example:

```
openGauss=# SELECT point(23.4, -44.5) AS RESULT;
    result
--------------
 (23.4,-44.5)
(1 row)
```

- point(box)

  Description: Center of a box

  Return type: point

  Example:

```
openGauss=# SELECT point(box '((-1,0),(1,0))') AS RESULT;
 result
--------
 (0,0)
(1 row)
```

- point(circle)

  Description: Center of a circle

  Return type: point

  Example:

```
openGauss=# SELECT point(circle '((0,0),2.0)') AS RESULT;
 result
--------
 (0,0)
(1 row)
```

- point(lseg)

  Description: Center of a line segment

  Return type: point

  Example:

```
openGauss=# SELECT point(lseg '((-1,0),(1,0))') AS RESULT;
 result
--------
 (0,0)
(1 row)
```

- point(polygon)

  Description: Center of a polygon

  Return type: point

  Example:

```
openGauss=# SELECT point(polygon '((0,0),(1,1),(2,0))') AS RESULT;
        result
-----------------------
 (1,0.333333333333333)
(1 row)
```

- polygon(box)

  Description: Box to 4-point polygon

  Return type: polygon

  Example:

```
openGauss=# SELECT polygon(box '((0,0),(1,1))') AS RESULT;
          result
---------------------------
 ((0,0),(0,1),(1,1),(1,0))
(1 row)
```

- polygon(circle)

  Description: Circle to 12-point polygon

  Return type: polygon

  Example:

```
openGauss=# SELECT polygon(circle '((0,0),2.0)') AS RESULT;

result

--------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------
 ((-2,0),(-1.73205080756888,1),(-1,1.73205080756888),(-1.22464679914735e-16,2),
(1,1.73205080756888),(1.73205080756888,1),(2,2.44929359829471e-16),
(1.73205080756888,-0.999999999999999),(1,-1.73205080756888),(3.67394039744206e-16,-2),
(-0.999999999999999,-1.73205080756888),(-1.73205080756888,-1))
(1 row)
```

- polygon(npts, circle)

  Description: Circle to **npts**-point polygon

  Return type: polygon

  Example:

```
openGauss=# SELECT polygon(12, circle '((0,0),2.0)') AS RESULT;

result

--------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------
 ((-2,0),(-1.73205080756888,1),(-1,1.73205080756888),(-1.22464679914735e-16,2),
(1,1.73205080756888),(1.73205080756888,1),(2,2.44929359829471e-16),
(1.73205080756888,-0.999999999999999),(1,-1.73205080756888),(3.67394039744206e-16,-2),
(-0.999999999999999,-1.73205080756888),(-1.73205080756888,-1))
(1 row)
```

- polygon(path)

  Description: Path to polygon

  Return type: polygon

  Example:

```
openGauss=# SELECT polygon(path '((0,0),(1,1),(2,0))') AS RESULT;
      result
---------------------
 ((0,0),(1,1),(2,0))
(1 row)
```

11.5.11 Network Address Functions and Operators

cidr and inet Operators

The operators **<<**, **<<=**, **>>**, and **>>=** test for subnet inclusion. They consider only the network parts of the two addresses (ignoring any host part) and determine whether one network is identical to or a subnet of the other.

- <

  Description: Is less than

  Example:

```
openGauss=# SELECT inet '192.168.1.5' < inet '192.168.1.6' AS RESULT;
 result
```

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

- <=

  Description: Is less than or equals

  Example:

```
openGauss=# SELECT inet '192.168.1.5' <= inet '192.168.1.5' AS RESULT;
 result
--------
 t
(1 row)
```

- =

  Description: Equals

  Example:

```
openGauss=# SELECT inet '192.168.1.5' = inet '192.168.1.5' AS RESULT;
 result
--------
 t
(1 row)
```

- >=

  Description: Is greater than or equals

  Example:

```
openGauss=# SELECT inet '192.168.1.5' >= inet '192.168.1.5' AS RESULT;
 result
--------
 t
(1 row)
```

- >

  Description: Is greater than

  Example:

```
openGauss=# SELECT inet '192.168.1.5' > inet '192.168.1.4' AS RESULT;
 result
--------
 t
(1 row)
```

- <>

  Description: Does not equal to

  Example:

```
openGauss=# SELECT inet '192.168.1.5' <> inet '192.168.1.4' AS RESULT;
 result
--------
 t
(1 row)
```

- <<

  Description: Is contained in

  Example:

```
openGauss=# SELECT inet '192.168.1.5' << inet '192.168.1/24' AS RESULT;
 result
--------
 t
(1 row)
```

- <<=

  Description: Is contained in or equals

Example:

```
openGauss=# SELECT inet '192.168.1/24' <<= inet '192.168.1/24' AS RESULT;
 result
--------
 t
(1 row)
```

- >>

  Description: Contains

  Example:

```
openGauss=# SELECT inet '192.168.1/24' >> inet '192.168.1.5' AS RESULT;
 result
--------
 t
(1 row)
```

- >>=

  Description: Contains or equals

  Example:

```
openGauss=# SELECT inet '192.168.1/24' >>= inet '192.168.1/24' AS RESULT;
 result
--------
 t
(1 row)
```

- ~

  Description: Bitwise NOT

  Example:

```
openGauss=# SELECT ~ inet '192.168.1.6' AS RESULT;
    result
---------------
 63.87.254.249
(1 row)
```

- &

  Description: Performs an AND operation on each bit of the two network addresses.

  Example:

```
openGauss=# SELECT inet '192.168.1.6' & inet '10.0.0.0' AS RESULT;
 result
---------
 0.0.0.0
(1 row)
```

- |

  Description: Performs an OR operation on each bit of the two network addresses.

  Example:

```
openGauss=# SELECT inet '192.168.1.6' | inet '10.0.0.0' AS RESULT;
   result
-------------
 202.168.1.6
(1 row)
```

- +

  Description: Addition

  Example:

```
openGauss=# SELECT inet '192.168.1.6' + 25 AS RESULT;
    result
```

```
--------------
 192.168.1.31
(1 row)
```

- -

  Description: Subtraction

  Example:

```
openGauss=# SELECT inet '192.168.1.43' - 36 AS RESULT;
   result
-------------
 192.168.1.7
(1 row)
```

- -

  Description: Subtraction

  Example:

```
openGauss=# SELECT inet '192.168.1.43' - inet '192.168.1.19' AS RESULT;
 result
--------
     24
(1 row)
```

cidr and inet Functions

The **abbrev**, **host**, and **text** functions are primarily intended to offer alternative display formats.

- abbrev(inet)

  Description: Abbreviated display format as text

  Return type: text

  Example:

```
openGauss=# SELECT abbrev(inet '10.1.0.0/16') AS RESULT;
   result
-------------
 10.1.0.0/16
(1 row)
```

- abbrev(cidr)

  Description: Abbreviated display format as text

  Return type: text

  Example:

```
openGauss=# SELECT abbrev(cidr '10.1.0.0/16') AS RESULT;
 result
---------
 10.1/16
(1 row)
```

- broadcast(inet)

  Description: Broadcast address for networks

  Return type: inet

  Example:

```
openGauss=# SELECT broadcast('192.168.1.5/24') AS RESULT;
      result
------------------
 192.168.1.255/24
(1 row)
```

- family(inet)

  Description: Extracts family of addresses, **4** for IPv4.

  Return type: int

  Example:

  ```
  openGauss=# SELECT family('127.0.0.1') AS RESULT;
   result
  --------
        4
  (1 row)
  ```

- host(inet)

  Description: Extracts IP addresses as text.

  Return type: text

  Example:

  ```
  openGauss=# SELECT host('192.168.1.5/24') AS RESULT;
     result
  -------------
   192.168.1.5
  (1 row)
  ```

- hostmask(inet)

  Description: Constructs the host mask for a network.

  Return type: inet

  Example:

  ```
  openGauss=# SELECT hostmask('192.168.23.20/30') AS RESULT;
   result
  ---------
   0.0.0.3
  (1 row)
  ```

- masklen(inet)

  Description: Extracts subnet mask length.

  Return type: int

  Example:

  ```
  openGauss=# SELECT masklen('192.168.1.5/24') AS RESULT;
   result
  --------
       24
  (1 row)
  ```

- netmask(inet)

  Description: Constructs the subnet mask for a network.

  Return type: inet

  Example:

  ```
  openGauss=# SELECT netmask('192.168.1.5/24') AS RESULT;
      result
  ---------------
   255.255.255.0
  (1 row)
  ```

- network(inet)

  Description: Extracts the network part of an address.

  Return type: cidr

  Example:

  ```
  openGauss=# SELECT network('192.168.1.5/24') AS RESULT;
       result
  ```

```
----------------
 192.168.1.0/24
(1 row)
```

- set_masklen(inet, int)

  Description: Sets subnet mask length for the **inet** value.

  Return type: inet

  Example:

```
openGauss=# SELECT set_masklen('192.168.1.5/24', 16) AS RESULT;
     result
----------------
 192.168.1.5/16
(1 row)
```

- set_masklen(cidr, int)

  Description: Sets subnet mask length for the **cidr** value.

  Return type: cidr

  Example:

```
openGauss=# SELECT set_masklen('192.168.1.0/24'::cidr, 16) AS RESULT;
     result
----------------
 192.168.0.0/16
(1 row)
```

- text(inet)

  Description: Extracts IP addresses and subnet mask length as text.

  Return type: text

  Example:

```
openGauss=# SELECT text(inet '192.168.1.5') AS RESULT;
     result
----------------
 192.168.1.5/32
(1 row)
```

Any **cidr** value can be cast to **inet** implicitly or explicitly; therefore, the functions shown above as operating on **inet** also work on **cidr** values. An **inet** value can be cast to **cidr**. After the conversion, any bits to the right of the subnet mask are silently zeroed to create a valid **cidr** value. In addition, you can cast a text string to **inet** or **cidr** using normal casting syntax. For example, **inet(expression)** or **colname::cidr**.

macaddr Functions

The function **trunc(macaddr)** returns a MAC address with the last 3 bytes set to zero.

- trunc(macaddr)

  Description: Sets last 3 bytes to zero.

  Return type: macaddr

  Example:

```
openGauss=# SELECT trunc(macaddr '12:34:56:78:90:ab') AS RESULT;
      result
-------------------
 12:34:56:00:00:00
(1 row)
```

The **macaddr** type also supports the standard relational operators (such as **>** and **<=**) for lexicographical ordering, and the bitwise arithmetic operators (**~**, **&** and **|**) for NOT, AND and OR.

11.5.12 Text Search Functions and Operators

Text Search Operators

- @@

  Description: Specifies whether the **tsvector**-type words match the **tsquery**-type words.

  Example:

```
openGauss=# SELECT to_tsvector('fat cats ate rats') @@ to_tsquery('cat & rat') AS RESULT;
 result
--------
 t
(1 row)
```

- @@@

  Description: Synonym for @@

  Example:

```
openGauss=# SELECT to_tsvector('fat cats ate rats') @@@ to_tsquery('cat & rat') AS RESULT;
 result
--------
 t
(1 row)
```

- ||

  Description: Connects two **tsvector**-type words.

  Example:

```
openGauss=# SELECT 'a:1 b:2'::tsvector || 'c:1 d:2 b:3'::tsvector AS RESULT;
         result
---------------------------
 'a':1 'b':2,5 'c':3 'd':4
(1 row)
```

- &&

  Description: Performs an AND operation on two **tsquery**-type words.

  Example:

```
openGauss=# SELECT 'fat | rat'::tsquery && 'cat'::tsquery AS RESULT;
         result
---------------------------
 ( 'fat' | 'rat' ) & 'cat'
(1 row)
```

- ||

  Description: Performs an OR operation on two **tsquery**-type words.

  Example:

```
openGauss=# SELECT 'fat | rat'::tsquery || 'cat'::tsquery AS RESULT;
         result
---------------------------
 ( 'fat' | 'rat' ) | 'cat'
(1 row)
```

- !!

  Description: **NOT** a **tsquery**

  Example:

```
openGauss=# SELECT !! 'cat'::tsquery AS RESULT;
 result
--------
 !'cat'
(1 row)
```

- @>

  Description: Specifies whether a **tsquery**-type word contains another **tsquery**-type word.

  Example:

```
openGauss=# SELECT 'cat'::tsquery @> 'cat & rat'::tsquery AS RESULT;
 result
--------
 f
(1 row)
```

- <@

  Description: Specifies whether a **tsquery**-type word is contained in another **tsquery**-type word.

  Example:

```
openGauss=# SELECT 'cat'::tsquery <@ 'cat & rat'::tsquery AS RESULT;
 result
--------
 t
(1 row)
```

In addition to the preceding operators, the ordinary B-tree comparison operators (including = and <) are defined for types **tsvector** and **tsquery**.

Text Search Functions

- get_current_ts_config()

  Description: Obtains default text search configurations.

  Return type: regconfig

  Example:

```
openGauss=# SELECT get_current_ts_config();
 get_current_ts_config
-----------------------
 english
(1 row)
```

- length(tsvector)

  Description: Specifies the number of lexemes in a **tsvector**-type word.

  Return type: integer

  Example:

```
openGauss=# SELECT length('fat:2,4 cat:3 rat:5A'::tsvector);
 length
--------
      3
(1 row)
```

- numnode(tsquery)

  Description: Specifies the number of lexemes plus **tsquery** operators.

  Return type: integer

  Example:

```
openGauss=# SELECT numnode('(fat & rat) | cat'::tsquery);
 numnode
---------
```

```
       5
(1 row)
```

- plainto_tsquery([config regconfig ,] query text)

  Description: Generates **tsquery** lexemes without punctuations.

  Return type: tsquery

  Example:

```
openGauss=# SELECT plainto_tsquery('english', 'The Fat Rats');
 plainto_tsquery
-----------------
 'fat' & 'rat'
(1 row)
```

- querytree(query tsquery)

  Description: Obtains the indexable part of a **tsquery**.

  Return type: text

  Example:

```
openGauss=# SELECT querytree('foo & ! bar'::tsquery);
 querytree
-----------
 'foo'
(1 row)
```

- setweight(tsvector, "char")

  Description: Assigns weight to each element of **tsvector**.

  Return type: tsvector

  Example:

```
openGauss=# SELECT setweight('fat:2,4 cat:3 rat:5B'::tsvector, 'A');
           setweight
-------------------------------
 'cat':3A 'fat':2A,4A 'rat':5A
(1 row)
```

- strip(tsvector)

  Description: Removes positions and weights from **tsvector**.

  Return type: tsvector

  Example:

```
openGauss=# SELECT strip('fat:2,4 cat:3 rat:5A'::tsvector);
       strip
-------------------
 'cat' 'fat' 'rat'
(1 row)
```

- to_tsquery([config regconfig ,] query text)

  Description: Normalizes words and converts them to **tsquery**.

  Return type: tsquery

  Example:

```
openGauss=# SELECT to_tsquery('english', 'The & Fat & Rats');
  to_tsquery
---------------
 'fat' & 'rat'
(1 row)
```

- to_tsvector([config regconfig ,] document text)

  Description: Reduces document text to **tsvector**.

  Return type: tsvector

  Example:

```
openGauss=# SELECT to_tsvector('english', 'The Fat Rats');
   to_tsvector
-----------------
 'fat':2 'rat':3
(1 row)
```

- to_tsvector_for_batch([config regconfig ,] document text)

  Description: Reduces document text to **tsvector**.

  Return type: tsvector

  Example:

```
openGauss=# SELECT to_tsvector_for_batch('english', 'The Fat Rats');
   to_tsvector
-----------------
 'fat':2 'rat':3
(1 row)
```

- ts_headline([config regconfig,] document text, query tsquery [, options text])

  Description: Highlights a query match.

  Return type: text

  Example:

```
openGauss=# SELECT ts_headline('x y z', 'z'::tsquery);
 ts_headline
--------------
 x y <b>z</b>
(1 row)
```

- ts_rank([weights float4[],] vector tsvector, query tsquery [, normalization integer])

  Description: Ranks documents for a query.

  Return type: float4

  Example:

```
openGauss=# SELECT ts_rank('hello world'::tsvector, 'world'::tsquery);
 ts_rank
----------
 .0607927
(1 row)
```

- ts_rank_cd([weights float4[],] vector tsvector, query tsquery [, normalization integer])

  Description: Ranks documents for a query using cover density.

  Return type: float4

  Example:

```
openGauss=# SELECT ts_rank_cd('hello world'::tsvector, 'world'::tsquery);
 ts_rank_cd
------------
          .0
(1 row)
```

- ts_rewrite(query tsquery, target tsquery, substitute tsquery)

  Description: Replaces a **tsquery**-type word.

  Return type: tsquery

  Example:

```
openGauss=# SELECT ts_rewrite('a & b'::tsquery, 'a'::tsquery, 'foo|bar'::tsquery);
       ts_rewrite
-------------------------
 'b' & ( 'foo' | 'bar' )
(1 row)
```

- ts_rewrite(query tsquery, select text)

  Description: Replaces **tsquery** data in the target with the result of a **SELECT** command.

  Return type: tsquery

  Example:

```
openGauss=# SELECT ts_rewrite('world'::tsquery, 'select ''world''::tsquery, ''hello''::tsquery');
 ts_rewrite
------------
 'hello'
(1 row)
```

Text Search Debugging Functions

- ts_debug([config regconfig,] document text, OUT alias text, OUT description text, OUT token text, OUT dictionaries regdictionary[], OUT dictionary regdictionary, OUT lexemes text[])

  Description: Tests a configuration.

  Return type: SETOF record

  Example:

```
openGauss=# SELECT ts_debug('english', 'The Brightest supernovaes');
                                 ts_debug
-----------------------------------------------------------------------------------
 (asciiword,"Word, all ASCII",The,{english_stem},english_stem,{})
 (blank,"Space symbols"," ",{},,)
 (asciiword,"Word, all ASCII",Brightest,{english_stem},english_stem,{brightest})
 (blank,"Space symbols"," ",{},,)
 (asciiword,"Word, all ASCII",supernovaes,{english_stem},english_stem,{supernova})
(5 rows)
```

- ts_lexize(dict regdictionary, token text)

  Description: Tests a data dictionary.

  Return type: text[]

  Example:

```
openGauss=# SELECT ts_lexize('english_stem', 'stars');
 ts_lexize
-----------
 {star}
(1 row)
```

- ts_parse(parser_name text, document text, OUT tokid integer, OUT token text)

  Description: Tests a parser.

  Return type: SETOF record

  Example:

```
openGauss=# SELECT ts_parse('default', 'foo - bar');
 ts_parse
-----------
 (1,foo)
 (12," ")
 (12,"- ")
 (1,bar)
(4 rows)
```

- ts_parse(parser_oid oid, document text, OUT tokid integer, OUT token text)

  Description: Tests a parser.

  Return type: SETOF record

Example:

```
openGauss=# SELECT ts_parse(3722, 'foo - bar');
 ts_parse
-----------
 (1,foo)
 (12," ")
 (12,"- ")
 (1,bar)
(4 rows)
```

- ts_token_type(parser_name text, OUT tokid integer, OUT alias text, OUT description text)

  Description: Obtains token types defined by a parser.

  Return type: SETOF record

  Example:

```
openGauss=# SELECT ts_token_type('default');
                     ts_token_type
--------------------------------------------------------------
 (1,asciiword,"Word, all ASCII")
 (2,word,"Word, all letters")
 (3,numword,"Word, letters and digits")
 (4,email,"Email address")
 (5,url,URL)
 (6,host,Host)
 (7,sfloat,"Scientific notation")
 (8,version,"Version number")
 (9,hword_numpart,"Hyphenated word part, letters and digits")
 (10,hword_part,"Hyphenated word part, all letters")
 (11,hword_asciipart,"Hyphenated word part, all ASCII")
 (12,blank,"Space symbols")
 (13,tag,"XML tag")
 (14,protocol,"Protocol head")
 (15,numhword,"Hyphenated word, letters and digits")
 (16,asciihword,"Hyphenated word, all ASCII")
 (17,hword,"Hyphenated word, all letters")
 (18,url_path,"URL path")
 (19,file,"File or path name")
 (20,float,"Decimal notation")
 (21,int,"Signed integer")
 (22,uint,"Unsigned integer")
 (23,entity,"XML entity")
(23 rows)
```

- ts_token_type(parser_oid oid, OUT tokid integer, OUT alias text, OUT description text)

  Description: Obtains token types defined by a parser.

  Return type: SETOF record

  Example:

```
openGauss=# SELECT ts_token_type(3722);
                     ts_token_type
--------------------------------------------------------------
 (1,asciiword,"Word, all ASCII")
 (2,word,"Word, all letters")
 (3,numword,"Word, letters and digits")
 (4,email,"Email address")
 (5,url,URL)
 (6,host,Host)
 (7,sfloat,"Scientific notation")
 (8,version,"Version number")
 (9,hword_numpart,"Hyphenated word part, letters and digits")
 (10,hword_part,"Hyphenated word part, all letters")
 (11,hword_asciipart,"Hyphenated word part, all ASCII")
 (12,blank,"Space symbols")
 (13,tag,"XML tag")
 (14,protocol,"Protocol head")
 (15,numhword,"Hyphenated word, letters and digits")
 (16,asciihword,"Hyphenated word, all ASCII")
 (17,hword,"Hyphenated word, all letters")
 (18,url_path,"URL path")
 (19,file,"File or path name")
 (20,float,"Decimal notation")
 (21,int,"Signed integer")
 (22,uint,"Unsigned integer")
 (23,entity,"XML entity")
(23 rows)
```

- ts_stat(sqlquery text, [weights text,] OUT word text, OUT ndoc integer, OUT nentry integer)

  Description: Obtains statistics of a **tsvector** column.

  Return type: SETOF record

  Example:

```
openGauss=# SELECT ts_stat('select ''hello world''::tsvector');
   ts_stat
-------------
 (world,1,1)
 (hello,1,1)
(2 rows)
```

11.5.13 JSON/JSONB Functions and Operators

For details about the JSON/JSONB data type, see **JSON/JSONB Types**.

Table 11-34 JSON/JSONB common operators

Operator	Left Operand Type	Right Operand Type	Return Type	Description	Example	Example Result
->	Array-json(b)	int	json(b)	Obtains the **array-json** element. If the subscript does not exist, **NULL** is returned.	'[{"a":"foo"}, {"b":"bar"}, {"c":"baz"}]'::json->2	{"c":"baz"}
->	object-json(b)	text	json(b)	Obtains the value by a key. If no record is found, **NULL** is returned.	'{"a": {"b":"foo"}}'::json->'a'	{"b":"foo"}

Operator	Left Operand Type	Right Operand Type	Return Type	Description	Example	Example Result
->>	Array-json(b)	int	text	Obtains the JSON array element. If the subscript does not exist, **NULL** is returned.	'[1,2,3]'::json->>2	3
->>	object-json(b)	text	text	Obtains the value by a key. If no record is found, **NULL** is returned.	'{"a":1,"b":2}'::json->>'b'	2
#>	container-json (b)	text[]	json(b)	Obtains the JSON object in the specified path. If the path does not exist, **NULL** is returned.	'{"a": {"b":{"c": "foo"}}}'::json #>'{a,b}'	{"c": "foo"}
#>>	container-json (b)	text[]	text	Obtains the JSON object in the specified path. If the path does not exist, **NULL** is returned.	'{"a": [1,2,3],"b": [4,5,6]}'::json #>>'{a,2}'	3

CAUTION

For the **#>** and **#>>** operators, if no data can be found in the specified path, no error is reported and a **NULL** value is returned.

Table 11-35 Additional JSONB support for operators

Operator	Right Operand Type	Description	Example
@>	jsonb	Whether the top layer of the JSON on the left contains all items of the top layer of the JSON on the right.	'{"a":1, "b":2}'::jsonb @> '{"b":2}'::jsonb
<@	jsonb	Whether all items in the JSON file on the left exist at the top layer of the JSON file on the right.	'{"b":2}'::jsonb <@ '{"a":1, "b":2}'::jsonb
?	text	Whether the string of the key or element exists at the top layer of the JSON value.	'{"a":1, "b":2}'::jsonb ? 'b'
?|	text[]	Whether any of these array strings exists as top-layer keys.	'{"a":1, "b":2, "c":3}'::jsonb ?| array['b', 'c']
?&	text[]	Whether all these array strings exist as top-layer keys.	'["a", "b"]'::jsonb ? & array['a', 'b']
=	jsonb	Determines the size between two **JSONB** files, which is the same as the **jsonb_eq** function.	/
<>	jsonb	Determines the size between two **JSONB** files, which is the same as the **jsonb_ne** function.	/

Operator	Right Operand Type	Description	Example
<	jsonb	Determines the size between two **JSONB** files, which is the same as the **jsonb_lt** function.	/
>	jsonb	Determines the size between two **JSONB** files, which is the same as the **jsonb_gt** function.	/
<=	jsonb	Determines the size between two **JSONB** files, which is the same as the **jsonb_le** function.	/
>=	jsonb	Determines the size between two **JSONB** files, which is the same as the **jsonb_ge** function.	/

Functions Supported by JSON/JSONB

- array_to_json\(anyarray \[, pretty_bool\]\)

  Description: Returns an array as JSON. A multi-dimensional array becomes a JSON array of arrays. If the value of **pretty_bool** is **true**, a newline character is added between one-dimensional elements.

  Return type: json

  For example:
```
openGauss=# SELECT array_to_json('{{1,5},{99,100}}'::int[]);
   array_to_json
   ------------------
   [[1,5],[99,100]]
   (1 row)
```

- row_to_json\(record \[, pretty_bool\]\)

  Description: Returns a row as JSON. If the value of **pretty_bool** is **true**, a newline character is added between one-dimensional elements.

  Return type: json

  For example:
```
openGauss=# SELECT row_to_json(row(1,'foo'));
       row_to_json
```

```
   ---------------------
    {"f1":1,"f2":"foo"}    (1 row)
```

- json_array_element(array-json, integer), jsonb_array_element(array-jsonb, integer)

  Description: Same as the operator `->`, which returns the element with the specified subscript in the array.

  Return type: json, jsonb

  For example:

```
openGauss=# select json_array_element('[1,true,[1,[2,3
]],null]',2);
  json_array_element
 --------------------
  [1,[2,3]]
  (1 row)
```

- json_array_element_text(array-json, integer), jsonb_array_element_text(array-jsonb, integer)

  Description: Same as the operator `->>`, which returns the element with the specified subscript in the array.

  Return type: text, text

  For example:

```
openGauss=# select json_array_element_text('[1,true,[1,[2,3]],null]',2);
json_array_element_text
-----------------------
  [1,[2,3]]
  (1 row)
```

- json_object_field(object-json, text), jsonb_object_field(object-jsonb, text)

  Description: Same as the operator `->`, which returns the value of a specified key in an object.

  Return type: json, json

  For example:

```
 openGauss=# select json_object_field('{"a": {"b":"foo"}}','a');
  json_object_field
  -------------------
  {"b":"foo"}
  (1 row)
```

- json_object_field_text(object-json, text), jsonb_object_field_text(object-jsonb, text)

  Description: Same as the operator `->`, which returns the value of a specified key in an object.

  Return type: text, text

  For example:

```
 openGauss=# select json_object_field_text('{"a": {"b":"foo"}}','a');
  json_object_field_text
  ----------------------
  {"b":"foo"}
  (1 row)
```

- json_extract_path(json, VARIADIC text[]), jsonb_extract_path((jsonb, VARIADIC text[])

  Description: Equivalent to the operator **`#>`** searches for JSON based on the path specified by *$2* and returns the result.

  Return type: json, jsonb

  For example:

```
openGauss=# select json_extract_path('{"f2":{"f3":1},"f4":{"f5":99,"f6":"stringy"}}', 'f4','f6');
  json_extract_path
-------------------
  "stringy"
 (1 row)
```

- json_extract_path_op(json, text[]), jsonb_extract_path_op(jsonb, text[])

  Description: Same as the operator **`#>`**, searches for JSON based on the path specified by *$2* and returns the result.

  Return type: json, jsonb

  For example:

```
 openGauss=# select json_extract_path_op('{"f2":{"f3":1},"f4":{"f5":99,"f6":"stringy"}}',
ARRAY['f4','f6']);
  json_extract_path_op
---------------------
  "stringy"
 (1 row)
```

- json_extract_path_text(json, VARIADIC text[]), jsonb_extract_path_text((jsonb, VARIADIC text[])

  Description: Equivalent to the operator **`#>`**, searches for JSON based on the path specified by $2 and return the result.

  Return type: text, text

  For example:

```
openGauss=# select json_extract_path_text('{"f2":{"f3":1},"f4":{"f5":99,"f6":"stringy"}}', 'f4','f6');
  json_extract_path_text
-----------------------
  "stringy"
 (1 row)
```

- json_extract_path_text_op(json, text[]), jsonb_extract_path_text_op(jsonb, text[])

  Description: Same as the operator **`#>`**, searches for JSON based on the path specified by *$2* and returns the result.

  Return type: text, text

  For example:

```
openGauss=# select json_extract_path_text_op('{"f2":{"f3":1},"f4":{"f5":99,"f6":"stringy"}}',
ARRAY['f4','f6']);
  json_extract_path_text_op
--------------------------
  "stringy"
 (1 row)
```

- Json_array_elements(array-json), jsonb_array_elements(array-jsonb)

  Description: Splits an array. Each element returns a row.

  Return type: json, jsonb

  For example:

```
openGauss=# select json_array_elements('[1,true,[1,[2,3]],null]');
  json_array_elements
---------------------
  1
  true
  [1,[2,3]]
  null
 (4 rows)
```

- Json_array_elements_text(array-json), jsonb_array_elements_text(array-jsonb)

  Description: Splits an array. Each element returns a row.

  Return type: text, text

For example:
```
openGauss=# select * from  json_array_elements_text('[1,true,[1,[2,3]],null]');
    value
 -----------
  1
  true
  [1,[2,3]]

 (4 rows)
```

- json_array_length(array-json), jsonb_array_length(array-jsonb)

  Description: Returns the array length.

  Return type: integer

  For example:
  ```
  openGauss=# SELECT json_array_length('[1,2,3,{"f1":1,"f2":[5,6]},4,null]');
   json_array_length
  -------------------
                  6
  (1 row)
  ```

- json_each(object-json), jsonb_each(object-jsonb)

  Description: Splits each key-value pair of an object into one row and two columns.

  Return type: setof(key text, value json), setof(key text, value jsonb)

  For example:
  ```
  openGauss=# select * from  json_each('{"f1":[1,2,3],"f2":{"f3":1},"f4":null}');
    key |  value
   -----+----------
    f1  | [1,2,3]
    f2  | {"f3":1}
    f4  | null
   (3 rows)
  ```

- json_each_text(object-json), jsonb_each_text(object-jsonb)

  Description: Splits each key-value pair of an object into one row and two columns.

  Return type: setof(key text, value text), setof(key text, value text)

  For example:
  ```
  openGauss=# select * from  json_each_text('{"f1":[1,2,3],"f2":{"f3":1},"f4":null}');
    key |  value
   -----+----------
    f1  | [1,2,3]
    f2  | {"f3":1}
    f4  |
   (3 rows)
  ```

- json_object_keys(object-json), jsonb_object_keys(object-jsonb)

  Description: Returns all keys at the top layer of the object.

  Return type: SETOF text

  For example:
  ```
  openGauss=# select json_object_keys('{"f1":"abc","f2":{"f3":"a", "f4":"b"}, "f1":"abcd"}');
   json_object_keys
  ------------------
   f1
   f2
   f1
  (3 rows)
  ```

- JSONB deduplication operations:
  ```
  openGauss=# select jsonb_object_keys('{"f1":"abc","f2":{"f3":"a", "f4":"b"}, "f1":"abcd"}');
   jsonb_object_keys
  -------------------
  ```

```
 f1
 f2
(2 rows)
```

- json_populate_record(anyelement, object-json [, bool]), jsonb_populate_record(anyelement, object-jsonb [, bool])

  Description: *$1* must be a compound parameter. Each key-value in the **object-json** file is split. The key is used as the column name to match the column name in *$1* and fill in the *$1* format.

  Return type: anyelement, anyelement

  For example:
```
openGauss=# create type jpop as (a text, b int, c bool);
CREATE TYPE
postgres=# select * from json_populate_record(null::jpop,'{"a":"blurfl","x":43.2}');
   a    | b | c
--------+---+---
 blurfl |   |
(1 row)
```
```
openGauss=# select * from json_populate_record((1,1,null)::jpop,'{"a":"blurfl","x":43.2}');
   a    | b | c
--------+---+---
 blurfl | 1 |
(1 row)
```

- json_populate_record_set(anyelement, array-json [, bool]), jsonb_populate_record_set(anyelement, array-jsonb [, bool])

  Description: Performs the preceding operations on each element in the *$2* array by referring to the **json_populate_record** and **jsonb_populate_record** functions. Therefore, each element in the *$2* array must be of the **object-json** type.

  Return type: setof anyelement, setof anyelement

  For example:
```
openGauss=# create type jpop as (a text, b int, c bool);
 CREATE TYPE
 postgres=# select * from json_populate_recordset(null::jpop, '[{"a":1,"b":2},{"a":3,"b":4}]');
  a | b | c
 ---+---+---
  1 | 2 |
  3 | 4 |
 (2 rows)
```

- json_typeof(json), jsonb_typeof(jsonb)

  Description: Checks the JSON type.

  Return type: text, text

  For example:
```
openGauss=# select value, json_typeof(value)
  postgres-# from (values (json '123.4'), (json '"foo"'), (json 'true'), (json 'null'), (json '[1, 2, 3]'), (json
'{"x":"foo", "y":123}'), (NULL::json))  as data(value);
        value         | json_typeof
----------------------+-------------
 123.4                | number
 "foo"                | string
 true                 | boolean
 null                 | null
 [1, 2, 3]            | array
 {"x":"foo", "y":123} | object
                      |
(7 rows)
```

- json_build_array([VARIADIC "any"])

  Description: Constructs a JSON array from a variable parameter list.

Return type: array-json

For example:

```
openGauss=# select json_build_array('a',1,'b',1.2,'c',true,'d',null,'e',json '{"x": 3, "y": [1,2,3]}','');
                         json_build_array
 ---------------------------------------------------------------------------
  ["a", 1, "b", 1.2, "c", true, "d", null, "e", {"x": 3, "y": [1,2,3]}, ""]
 (1 row)
```

- json_build_object([VARIADIC "any"])

  Description: Constructs a JSON object from a variable parameter list. The number of input parameters must be an even number. Every two input parameters form a key-value pair. Note that the value of a key cannot be null.

  Return type: object-json

  For example:

```
openGauss=# select json_build_object(1,2);
  json_build_object
 -------------------
  {"1" : 2}
 (1 row)
```

- json_to_record(object-json, bool)

  Description: Like all functions that return **record**, the caller must explicitly define the structure of the record with an AS clause. The key-value pair of **object-json** is split and reassembled. The key is used as a column name to match and fill in the structure of the specified record.

  Return type: record

  For example:

```
openGauss=# select * from json_to_record('{"a":1,"b":"foo","c":"bar"}',true) as x(a int, b text, d text);
  a |  b  | d
 ---+-----+---
  1 | foo |
 (1 row)
```

- json_to_recordset(array-json, bool)

  Description: Executes the preceding function on each element in the array by referring to the **json_to_record** function. Therefore, each element in the array must be **object-json**.

  Return type: SETOF record

  For example:

```
openGauss=# select * from json_to_recordset(
 openGauss(#   '[{"a":1,"b":"foo","d":false},{"a":2,"b":"bar","c":true}]',
openGauss(#   false
openGauss(# ) as x(a int, b text, c boolean);
  a |  b  | c
 ---+-----+---
  1 | foo |
  2 | bar | t
 (2 rows)
```

- json_object(text[]), json_object(text[], text[])

  Description: Constructs an **object-json** from a text array. This is an overloaded function. When the input parameter is a text array, the array length must be an even number, and members are considered as alternate key-value pairs. When two text arrays are used, the first array is considered as a key, and the second array a value. The lengths of the two arrays must be the same. Note that the value of a key cannot be null.

  Return type: object-json

  For example:

```
openGauss=# select json_object('{a,1,b,2,3,NULL,"d e f","a b c"}');
                      json_object
-------------------------------------------------------
 {"a" : "1", "b" : "2", "3" : null, "d e f" : "a b c"}
(1 row)
postgres=# select json_object('{a,b,"a b c"}', '{a,1,1}');
              json_object
---------------------------------------
 {"a" : "a", "b" : "1", "a b c" : "1"}
(1 row)
```

- json_agg(any)

  Description: Aggregates values into a JSON array.

  Return type: array-json

  For example:

```
openGauss=# select * from classes;
 name | score
-----+-------
 A    |     2
 A    |     3
 D    |     5
 D    |
(4 rows)
 openGauss=# select name, json_agg(score) score from classes group by name order by name;
 name |      score
-----+-----------------
 A    | [2, 3]
 D    | [5, null]
      | [null]
(3 rows)
```

- json_object_agg(any, any)

  Description: Aggregates values into a JSON object.

  Return type: object-json

  For example:

```
openGauss=# select * from classes;
name | score
-----+-------
A    |     2
A    |     3
D    |     5
D    |
(4 rows)
```

```
openGauss=# select json_object_agg(name, score) from classes group by name order by name;
     json_object_agg
-------------------------
 { "A" : 2, "A" : 3 }
 { "D" : 5, "D" : null }
(2 rows)
```

- - jsonb_contained(jsonb, jsonb)

  Description: Same as the operator **`<@`**, determines whether all elements in *$1* exist at the top layer of *$2*.

  Return type: Boolean

  For example:

```
openGauss=# select jsonb_contained('[1,2,3]', '[1,2,3,4]');
  jsonb_contained
-----------------
  t
 (1 row)
```

- - jsonb_contains(jsonb, jsonb)

  Description: Same as the operator **`@>`**, checks whether all top-layer elements in *$1* are contained in *$2*.

Return type: Boolean

For example:

```
openGauss=# select jsonb_contains('[1,2,3,4]', '[1,2,3]');
 jsonb_contains
----------------
 t
(1 row)
```

- - jsonb_exists(jsonb, text)

  Description: Same as the operator **`?`**, determines whether all elements in the string array *$2* exist at the top layer of *$1* in the form of **key\elem\scalar**.

  Return type: Boolean

  For example:

  ```
  openGauss=# select jsonb_exists('["1",2,3]', '1');
   jsonb_exists
  --------------
   t
  (1 row)
  ```

- - jsonb_exists_all(jsonb, text[])

  Description: Same as the operator **`?&`**, checks whether all elements in the string array *$2* exist at the top layer of *$1* in the form of **key\elem\scalar**.

  Return type: Boolean

  For example:

  ```
  openGauss=# select jsonb_exists_all('["1","2",3]', '{1, 2}');
   jsonb_exists_all
  ------------------
   t
  (1 row)
  ```

- - jsonb_exists_any(jsonb, text[])

  Description: Same as the operator **`?|`**, checks whether all elements in the string array *$2* exist at the top layer of *$1* in the form of **key\elem\scalar**.

  Return type: Boolean

  For example:

  ```
  openGauss=# select jsonb_exists_any('["1","2",3]', '{1, 2, 4}');
   jsonb_exists_any
  ------------------
   t
  (1 row)
  ```

- - jsonb_cmp(jsonb, jsonb)

  Description: Compares values. A positive value indicates greater than, a negative value indicates less than, and **0** indicates equal.

  Return type: integer

  For example:

  ```
  openGauss=# select jsonb_cmp('["a", "b"]', '{"a":1, "b":2}');
   jsonb_cmp
  -----------
         -1
  (1 row)
  ```

- - jsonb_eq(jsonb, jsonb)

  Description: Same as the operator **`=`**, compares two values.

  Return type: Boolean

  For example:

  ```
  openGauss=# select jsonb_eq('["a", "b"]', '{"a":1, "b":2}');
   jsonb_eq
  ```

```
----------
 f
(1 row)
```

- - jsonb_ne(jsonb, jsonb)

  Description: Same as the operator **`<>`**, compares two values.

  Return type: Boolean

  For example:

```
openGauss=# select jsonb_ne('["a", "b"]', '{"a":1, "b":2}');
 jsonb_ne
----------
 t
(1 row)
```

- - jsonb_gt(jsonb, jsonb)

  Description: Same as the operator **`>`**, compares two values.

  Return type: Boolean

  For example:

```
openGauss=# select jsonb_gt('["a", "b"]', '{"a":1, "b":2}');
 jsonb_gt
----------
 f
(1 row)
```

- - jsonb_ge(jsonb, jsonb)

  Description: Same as the operator **`>=`**, compares two values.

  Return type: Boolean

  For example:

```
openGauss=# select jsonb_ge('["a", "b"]', '{"a":1, "b":2}');
 jsonb_ge
----------
 f
(1 row)
```

- - jsonb_lt(jsonb, jsonb)

  Description: Same as the operator **`<`**, compares two values.

  Return type: Boolean

  For example:

```
openGauss=# select jsonb_lt('["a", "b"]', '{"a":1, "b":2}');
 jsonb_lt
----------
 t
(1 row)
```

- - jsonb_le(jsonb, jsonb)

  Description: Same as the operator **`<=`**, compares two values.

  Return type: Boolean

  For example:

```
openGauss=# select jsonb_le('["a", "b"]', '{"a":1, "b":2}');
 jsonb_le
----------
 t
(1 row)
```

- - to_json(anyelement)

  Description: Converts parameters to **`json`**.

  Return type: json

  For example:

```
openGauss=# select to_json('{1,5}'::text[]);
   to_json
 -----------
  ["1","5"]
 (1 row)
```

- - jsonb_hash(jsonb)

  Description: Performs the hash operation on JSONB.

  Return type: integer

  For example:
```
openGauss=# select jsonb_hash('[1,2,3]');
 jsonb_hash
------------
 -559968547
(1 row)
```

- Other functions

  Description: Internal functions used by GIN indexes and JSON\JSONB aggregate functions.
```
 gin_compare_jsonb
 gin_consistent_jsonb
 gin_consistent_jsonb_hash
 gin_extract_jsonb
 gin_extract_jsonb_hash
 gin_extract_jsonb_query
 gin_extract_jsonb_query_hash
 gin_triconsistent_jsonb
 gin_triconsistent_jsonb_hash

 json_agg_transfn
 json_agg_finalfn
 json_object_agg_transfn
 json_object_agg_finalfn
```

11.5.14 HLL Functions and Operators

Hash Functions

- hll_hash_boolean(bool)

  Description: Hashes data of the Boolean type.

  Return type: hll_hashval

  Example:
```
openGauss=# SELECT hll_hash_boolean(FALSE);
  hll_hash_boolean
---------------------
 -5451962507482445012
(1 row)
```

- hll_hash_boolean(bool, int32)

  Description: Configures a hash seed (that is, change the hash policy) and hashes data of the bool type.

  Return type: hll_hashval

  Example:
```
openGauss=# SELECT hll_hash_boolean(FALSE, 10);
  hll_hash_boolean
--------------------
 -1169037589280886076
(1 row)
```

- hll_hash_smallint(smallint)

Description: Hashes data of the smallint type.

Return type: hll_hashval

Example:

```
openGauss=# SELECT hll_hash_smallint(100::smallint);
  hll_hash_smallint
---------------------
 962727970174027904
(1 row)
```

NOTE

If parameters with the same numeric value are hashed using different data types, the data will differ, because hash functions select different calculation policies for each type.

- hll_hash_smallint(smallint, int32)

  Description: Configures a hash seed (that is, change the hash policy) and hashes data of the smallint type.

  Return type: hll_hashval

  Example:

```
openGauss=# SELECT hll_hash_smallint(100::smallint, 10);
  hll_hash_smallint
---------------------
 -9056177146160443041
(1 row)
```

- hll_hash_integer(integer)

  Description: Hashes data of the integer type.

  Return type: hll_hashval

  Example:

```
openGauss=# SELECT hll_hash_integer(0);
   hll_hash_integer
----------------------
 5156626420896634997
(1 row)
```

- hll_hash_integer(integer, int32)

  Description: Hashes data of the integer type and configures a hash seed (that is, change the hash policy).

  Return type: hll_hashval

  Example:

```
openGauss=# SELECT hll_hash_integer(0, 10);
  hll_hash_integer
--------------------
 -5035020264353794276
(1 row)
```

- hll_hash_bigint(bigint)

  Description: Hashes data of the bigint type.

  Return type: hll_hashval

  Example:

```
openGauss=# SELECT hll_hash_bigint(100::bigint);
   hll_hash_bigint
---------------------
 -2401963681423227794
(1 row)
```

- hll_hash_bigint(bigint, int32)

Description: Hashes data of the bigint type and configures a hash seed (that is, change the hash policy).

Return type: hll_hashval

Example:

```
openGauss=# SELECT hll_hash_bigint(100::bigint, 10);
   hll_hash_bigint
---------------------
 -2305749404374433531
(1 row)
```

- hll_hash_bytea(bytea)

  Description: Hashes data of the bytea type.

  Return type: hll_hashval

  Example:

```
openGauss=# SELECT hll_hash_bytea(E'\\x');
 hll_hash_bytea
----------------
 0
(1 row)
```

- hll_hash_bytea(bytea, int32)

  Description: Hashes data of the bytea type and configures a hash seed (that is, change the hash policy).

  Return type: hll_hashval

  Example:

```
openGauss=# SELECT hll_hash_bytea(E'\\x', 10);
   hll_hash_bytea
---------------------
 7233188113542599437
(1 row)
```

- hll_hash_text(text)

  Description: Hashes data of the text type.

  Return type: hll_hashval

  Example:

```
openGauss=# SELECT hll_hash_text('AB');
    hll_hash_text
---------------------
 -5666002586880275174
(1 row)
```

- hll_hash_text(text, int32)

  Description: Hashes data of the text type and configures a hash seed (that is, change the hash policy).

  Return type: hll_hashval

  Example:

```
openGauss=# SELECT hll_hash_text('AB', 10);
hll_hash_text
---------------------
-2215507121143724132
(1 row)
```

- hll_hash_any(anytype)

  Description: Hashes data of any type.

  Return type: hll_hashval

  Example:

```
openGauss=# select hll_hash_any(1);
     hll_hash_any
----------------------
 -1316670585935156930
(1 row)

openGauss=# select hll_hash_any('08:00:2b:01:02:03'::macaddr);
     hll_hash_any
----------------------
 -3719950434455589360
(1 row)
```

- hll_hash_any(anytype, int32)

  Description: Hashes data of any type and configures a hash seed (that is, change the hash policy).

  Return type: hll_hashval

  Example:

```
openGauss=# select hll_hash_any(1, 10);
     hll_hash_any
----------------------
 7048553517657992351
(1 row)
```

- hll_hashval_eq(hll_hashval, hll_hashval)

  Description: Compares two pieces of data of the **hll_hashval** type to check whether they are the same.

  Return type: Boolean

  Example:

```
openGauss=# select hll_hashval_eq(hll_hash_integer(1), hll_hash_integer(1));
 hll_hashval_eq
----------------
 t
(1 row)
```

- hll_hashval_ne(hll_hashval, hll_hashval)

  Description: Compares two pieces of data of the **hll_hashval** type to check whether they are different.

  Return type: Boolean

  Example:

```
openGauss=# select hll_hashval_ne(hll_hash_integer(1), hll_hash_integer(1));
 hll_hashval_ne
----------------
 f
(1 row)
```

HLL Functions

There are three HLL modes: explicit, sparse, and full. When the data size is small, the explicit mode is used. In this mode, distinct values are calculated without errors. As the number of distinct values increases, the HLL mode is switched to the sparse and full modes in sequence. The two modes have no difference in the calculation result, but vary in the calculation efficiency of HLL functions and the storage space of HLL objects. The following functions can be used to view some HLL parameters:

- hll_print(hll)

  Description: Prints some debugging parameters of an HLL.

  Example:

```
openGauss=# select hll_print(hll_empty());
                                  hll_print
-------------------------------------------------------------------------------
 type=1(HLL_EMPTY), log2m=14, log2explicit=10, log2sparse=12, duplicatecheck=0
(1 row)
```

- hll_type(hll)

  Description: Checks the type of the current HLL. The return values are described as follows: **0** indicates **HLL_UNINIT**, an HLL object that is not initialized. **1** indicates **HLL_EMPTY**, an empty HLL object. **2** indicates **HLL_EXPLICIT**, an HLL object in explicit mode. **3** indicates **HLL_SPARSE**, an HLL object in sparse mode. **4** indicates **HLL_FULL**, an HLL object in full mode. **5** indicates **HLL_UNDEFINED**, an invalid HLL object.

  Example:

```
openGauss=# select hll_type(hll_empty());
 hll_type
----------
        1
(1 row)
```

- hll_log2m(hll)

  Description: Checks the value of **log2m** in the current HLL data structure. **log2m** is the logarithm of the number of buckets. This value affects the error rate of calculating distinct values by HLL. The error rate = ±1.04/√(2^log2m). If the value of **log2m** ranges from 10 to 16, HLL sets the number of buckets to 2^{log2m}. When the value of **log2explicit** is explicitly set to **–1**, the built-in default value is used.

  Example:

```
openGauss=# select hll_log2m(hll_empty());
 hll_log2m
-----------
        14
(1 row)

openGauss=# select hll_log2m(hll_empty(10));
 hll_log2m
-----------
        10
(1 row)

openGauss=# select hll_log2m(hll_empty(-1));
 hll_log2m
-----------
        14
(1 row)
```

- hll_log2explicit(hll)

  Description: Queries the **log2explicit** value in the current HLL data structure. Generally, the HLL changes from the explicit mode to the sparse mode and then to the full mode. This process is called the promotion hierarchy policy. You can change the value of **log2explicit** to change the policy. For example, if **log2explicit** is set to **0**, an HLL will skip the explicit mode and directly enter the sparse mode. When the value of **log2explicit** is explicitly set to a value ranging from 1 to 12, HLL will switch to the sparse mode when the length of the data segment exceeds $2^{log2explicit}$. When the value of **log2explicit** is explicitly set to **–1**, the built-in default value is used.

  Example:

```
openGauss=# select hll_log2explicit(hll_empty());
 hll_log2explicit
```

```
------------------
             10
(1 row)

openGauss=# select hll_log2explicit(hll_empty(12, 8));
 hll_log2explicit
------------------
              8
(1 row)

openGauss=# select hll_log2explicit(hll_empty(12, -1));
 hll_log2explicit
------------------
             10
(1 row)
```

- hll_log2sparse(hll)

  Description: Queries the value of **log2sparse** in the current HLL data structure. Generally, the HLL changes from the explicit mode to the sparse mode and then to the full mode. This process is called the promotion hierarchy policy. You can adjust the value of **log2sparse** to change the policy. For example, if the value of **log2sparse** is **0**, the system skips the sparse mode and directly enters the full mode. If the value of **log2sparse** is explicitly set to a value ranging from 1 to 14, HLL will switch to the full mode when the length of the data segment exceeds $2^{log2sparse}$. When the value of **log2sparse** is explicitly set to **–1**, the built-in default value is used.

  Example:

```
openGauss=# select hll_log2sparse(hll_empty());
 hll_log2sparse
----------------
            12
(1 row)

openGauss=# select hll_log2sparse(hll_empty(12, 8, 10));
 hll_log2sparse
----------------
            10
(1 row)

openGauss=# select hll_log2sparse(hll_empty(12, 8, -1));
 hll_log2sparse
----------------
            12
(1 row)
```

- hll_duplicatecheck(hll)

  Description: Specifies whether duplicate check is enabled. The value **0** indicates that it is disabled and the value **1** indicates that it is enabled. This function is disabled by default. If there are many duplicate values, you can enable this function to improve efficiency. When the value of **duplicatecheck** is explicitly set to **–1**, the built-in default value is used.

  Example:

```
openGauss=# select hll_duplicatecheck(hll_empty());
 hll_duplicatecheck
--------------------
                  0
(1 row)

openGauss=# select hll_duplicatecheck(hll_empty(12, 8, 10, 1));
 hll_duplicatecheck
--------------------
                  1
(1 row)
```

```
openGauss=# select hll_duplicatecheck(hll_empty(12, 8, 10, -1));
 hll_duplicatecheck
--------------------
                 0
(1 row)
```

Functional Functions

- hll_empty()

  Description: Creates an empty HLL.

  Return type: hll

  Example:

```
openGauss=# select hll_empty();
                        hll_empty
------------------------------------------------------------
 \x484c4c00000000002b05000000000000000000000000000000000000
(1 row)
```

- hll_empty(int32 log2m)

  Description: Creates an empty HLL and sets the **log2m** parameter. The parameter value ranges from 10 to 16. If the input is **–1**, the built-in default value is used.

  Return type: HLL

  Example:

```
openGauss=# select hll_empty(10);
                        hll_empty
------------------------------------------------------------
 \x484c4c00000000002b04000000000000000000000000000000000000
(1 row)

openGauss=# select hll_empty(-1);
                        hll_empty
------------------------------------------------------------
 \x484c4c00000000002b05000000000000000000000000000000000000
(1 row)
```

- hll_empty(int32 log2m, int32 log2explicit)

  Description: Creates an empty HLL and sets the **log2m** and **log2explicit** parameters in sequence. The value of **log2explicit** ranges from 0 to 12. The value **0** indicates that the explicit mode is skipped. This parameter is used to set the threshold of the explicit mode. When the length of the data segment reaches $2^{log2explicit}$, the mode is switched to the sparse or full mode. If the input is **–1**, the built-in default value of **log2explicit** is used.

  Return type: HLL

  Example:

```
openGauss=# select hll_empty(10, 4);
                        hll_empty
------------------------------------------------------------
 \x484c4c00000000001304000000000000000000000000000000000000
(1 row)

openGauss=# select hll_empty(10, -1);
                        hll_empty
------------------------------------------------------------
 \x484c4c00000000002b04000000000000000000000000000000000000
(1 row)
```

- hll_empty(int32 log2m, int32 log2explicit, int64 log2sparse)

Description: Creates an empty HLL and sets the **log2m**, **log2explicit** and **log2sparse** parameters in sequence. The value of **log2sparse** ranges from 0 to 14. The value **0** indicates that the sparse mode is skipped. This parameter is used to set the threshold of the sparse mode. When the length of the data segment reaches $2^{log2sparse}$, the mode is switched to the full mode. If the input is **–1**, the built-in default value of **log2sparse** is used.

Return type: HLL

Example:

```
openGauss=# select hll_empty(10, 4, 8);
                        hll_empty
------------------------------------------------------------
 \x484c4c00000000001204000000000000000000000000000000000000
(1 row)

openGauss=# select hll_empty(10, 4, -1);
                        hll_empty
------------------------------------------------------------
 \x484c4c00000000001304000000000000000000000000000000000000
(1 row)
```

- hll_empty(int32 log2m, int32 log2explicit, int64 log2sparse, int32 duplicatecheck)

  Description: Creates an empty HLL and sets the **log2m**, **log2explicit**, **log2sparse**, and **duplicatecheck** parameters in sequence. The value of **duplicatecheck** is **0** or **1**, indicating whether the duplicate check mode is enabled. By default, this mode is disabled. If the input is **–1**, the built-in default value of **duplicatecheck** is used.

  Return type: HLL

  Example:

```
openGauss=# select hll_empty(10, 4, 8, 0);
                        hll_empty
------------------------------------------------------------
 \x484c4c00000000001204000000000000000000000000000000000000
(1 row)

openGauss=# select hll_empty(10, 4, 8, -1);
                        hll_empty
------------------------------------------------------------
 \x484c4c00000000001204000000000000000000000000000000000000
(1 row)
```

- hll_add(hll, hll_hashval)

  Description: Adds **hll_hashval** to an HLL.

  Return type: HLL

  Example:

```
openGauss=# select hll_add(hll_empty(), hll_hash_integer(1));
                                  hll_add
----------------------------------------------------------------------------
 \x484c4c08000002002b0900000000000000f03f3e2921ff133fbaed3e2921ff133fbaed00
(1 row)
```

- hll_add_rev(hll_hashval, hll)

  Description: Adds **hll_hashval** to an HLL. This function works the same as **hll_add**, except that the positions of parameters are switched.

  Return type: HLL

  Example:

```
openGauss=# select hll_add_rev(hll_hash_integer(1), hll_empty());
                                hll_add_rev
```

```
----------------------------------------------------------------------------
 \x484c4c08000002002b0900000000000000f03f3e2921ff133fbaed3e2921ff133fbaed00
(1 row)
```

- hll_eq(hll, hll)

  Description: Compares two HLLs to check whether they are the same.

  Return type: Boolean

  Example:

```
openGauss=# select hll_eq(hll_add(hll_empty(), hll_hash_integer(1)), hll_add(hll_empty(),
hll_hash_integer(2)));
 hll_eq
--------
 f
(1 row)
```

- hll_ne(hll, hll)

  Description: Compares two HLLs to check whether they are different.

  Return type: Boolean

  Example:

```
openGauss=# select hll_ne(hll_add(hll_empty(), hll_hash_integer(1)), hll_add(hll_empty(),
hll_hash_integer(2)));
 hll_ne
--------
 t
(1 row)
```

- hll_cardinality(hll)

  Description: Calculates the number of distinct values of an HLL.

  Return type: int

  Example:

```
openGauss=# select hll_cardinality(hll_empty() || hll_hash_integer(1));
 hll_cardinality
-----------------
               1
(1 row)
```

- hll_union(hll, hll)

  Description: Performs an UNION operation on two HLL data structures to obtain one HLL.

  Return type: HLL

  Example:

```
openGauss=# select hll_union(hll_add(hll_empty(), hll_hash_integer(1)), hll_add(hll_empty(),
hll_hash_integer(2)));
                                        hll_union
--------------------------------------------------------------------------------------------

\x484c4c10002000002b090000000000000000400000000000000000b3ccc49320cca1ae3e2921ff133fba
ed00
(1 row)
```

Aggregate Functions

- hll_add_agg(hll_hashval)

  Description: Groups hashed data into HLL

  Return type: HLL

  Example:

```
-- Prepare data.
openGauss=# create table t_id(id int);
```

```
openGauss=# insert into t_id values(generate_series(1,500));
openGauss=# create table t_data(a int, c text);
openGauss=# insert into t_data select mod(id,2), id from t_id;

-- Create a table and specify an HLL column.
openGauss=# create table t_a_c_hll(a int, c hll);

-- Use GROUP BY on column a to group data, and insert the data to the HLL.
openGauss=# insert into t_a_c_hll select a, hll_add_agg(hll_hash_text(c)) from t_data group by a;

-- Calculate the number of distinct values for each group in the HLL.
openGauss=# select a, #c as cardinality from t_a_c_hll order by a;
 a |   cardinality
---+------------------
 0 | 247.862354346299
 1 | 250.908710610377
(2 rows)
```

- hll_add_agg(hll_hashval, int32 log2m)

  Description: Groups hashed data into HLL and specifies the **log2m** parameter. The value ranges from 10 to 16. If the input is **–1** or **NULL**, the built-in default value is used.

  Return type: HLL

  Example:

```
openGauss=# select hll_cardinality(hll_add_agg(hll_hash_text(c), 12)) from t_data;
 hll_cardinality
------------------
 497.965240179228
(1 row)
```

- hll_add_agg(hll_hashval, int32 log2m, int32 log2explicit)

  Description: Groups hashed data into HLL and specifies the **log2m** and **log2explicit** parameters in sequence. The value of **log2explicit** ranges from 0 to 12. The value **0** indicates that the explicit mode is skipped. This parameter is used to set the threshold of the explicit mode. When the length of the data segment reaches $2^{log2explicit}$, the mode is switched to the sparse or full mode. If the input is **–1** or **NULL**, the built-in default value of **log2explicit** is used.

  Return type: HLL

  Example:

```
openGauss=# select hll_cardinality(hll_add_agg(hll_hash_text(c), NULL, 1)) from t_data;
 hll_cardinality
------------------
 498.496062953313
(1 row)
```

- hll_add_agg(hll_hashval, int32 log2m, int32 log2explicit, int64 log2sparse)

  Description: Groups hashed data into HLL and sets the parameters **log2m**, **log2explicit**, and **log2sparse** in sequence. The value of **log2sparse** ranges from 0 to 14. The value **0** indicates that the sparse mode is skipped. This parameter is used to set the threshold of the sparse mode. When the length of the data segment reaches $2^{log2sparse}$, the mode is switched to the full mode. If the input is **–1** or **NULL**, the built-in default value of **log2sparse** is used.

  Return type: HLL

  Example:

```
openGauss=# select hll_cardinality(hll_add_agg(hll_hash_text(c), NULL, 6, 10)) from t_data;
 hll_cardinality
------------------
 498.496062953313
(1 row)
```

- hll_add_agg(hll_hashval, int32 log2m, int32 log2explicit, int64 log2sparse, int32 duplicatecheck)

  Description: Groups hashed data into HLL and sets the **log2m**, **log2explicit**, **log2sparse**, and **duplicatecheck** parameters. The value of **duplicatecheck** can be **0** or **1**, indicating whether to enable this mode. By default, this mode is disabled. If the input is **–1** or **NULL**, the built-in default value of **duplicatecheck** is used.

  Return type: HLL

  Example:

```
openGauss=# select hll_cardinality(hll_add_agg(hll_hash_text(c), NULL, 6, 10, -1)) from t_data;
 hll_cardinality
------------------
 498.496062953313
(1 row)
```

- hll_union_agg(hll)

  Description: Performs an UNION operation on multiple pieces of data of the HLL type to obtain one HLL.

  Return type: HLL

  Example:

```
-- Perform the UNION operation on data of the HLL type in each group to obtain one HLL, and
calculate the number of distinct values:
openGauss=# select #hll_union_agg(c) as cardinality from t_a_c_hll;
   cardinality
------------------
 498.496062953313
(1 row)
```

  NOTE

  To perform the UNION operation on data in multiple HLLs, ensure that the HLLs have the same precision. Otherwise, **UNION** cannot be performed. This constraint also applies to the **hll_union(hll, hll)** function.

Obsolete Functions

Some old HLL functions are discarded due to version upgrade. You can replace them with similar functions.

- hll_schema_version(hll)

  Description: Checks the schema version in the current HLL. In earlier versions, the schema version is fixed at **1**, which is used to verify the header of the HLL field. After refactoring, the HLL field is added to the header for verification. The schema version is no longer used.

- hll_regwidth(hll)

  Description: Queries the bucket size in the HLL data structure. In earlier versions, the value of **regwidth** ranges from 1 to 5, which has a large error and limits the upper limit of the cardinality estimation. After refactoring, the value of **regwidth** is fixed at **6** and the **regwidth** variable is not used.

- hll_expthresh(hll)

  Description: Obtains the **expthresh** value in the current HLL. The **hll_log2explicit(hll)** function is used to replace similar functions.

- hll_sparseon(hll)

Description: Specifies whether to enable the sparse mode. Use **hll_log2sparse(hll)** to replace similar functions. The value **0** indicates that the sparse mode is disabled.

Built-in Functions

HyperLogLog (HLL) has a series of built-in functions for internal data processing. Generally, users do not need to know how to use these functions. For details, see **Table 11-36**.

Table 11-36 Built-in Functions

Function	Function
hll_in	Receives hll data in string format.
hll_out	Sends hll data in string format.
hll_recv	Receives hll data in bytea format.
hll_send	Sends hll data in bytea format.
hll_trans_in	Receives hll_trans_type data in string format.
hll_trans_out	Sends hll_trans_type data in string format.
hll_trans_recv	Receives hll_trans_type data in bytea format.
hll_trans_send	Sends hll_trans_type data in bytea format.
hll_typmod_in	Receives typmod data.
hll_typmod_out	Sends typmod data.
hll_hashval_in	Receives hll_hashval data.
hll_hashval_out	Sends hll_hashval data.
hll_add_trans0	It is similar to **hll_add**. No input parameter is specified during initialization. It is usually used in the first phase of DNs in aggregation operations.
hll_add_trans1	It is similar to **hll_add**. An input parameter is specified during initialization. It is usually used in the first phase of DNs in aggregation operations.
hll_add_trans2	It is similar to **hll_add**. Two input parameters are specified during initialization. It is usually used in the first phase of DNs in aggregation operations.
hll_add_trans3	It is similar to **hll_add**. Three input parameters are specified during initialization. It is usually used in the first phase of DNs in aggregation operations.
hll_add_trans4	It is similar to **hll_add**. Four input parameters are specified during initialization. It is usually used in the first phase of DNs in aggregation operations.

Function	Function
hll_union_trans	It is similar to **hll_union** and is used in the first phase of DNs in aggregation operations.
hll_union_collect	It is similar to **hll_union** and is used in the second phase of DNs in aggregation operations to summarize the results of each DN.
hll_pack	It is used in the third phase of DNs in aggregation operations to convert a user-defined type hll_trans_type to the hll type.
hll	Converts an HLL type to another HLL type. Input parameters can be specified.
hll_hashval	Converts the bigint type to the **hll_hashval** type.
hll_hashval_int4	Converts the int4 type to the **hll_hashval** type.

Operators

- =

  Description: Compares the values of HLL and **hll_hashval** types to check whether they are the same.

  Return type: Boolean

  Example:

```
--hll
openGauss=# select (hll_empty() || hll_hash_integer(1)) = (hll_empty() || hll_hash_integer(1));
column
----------
 t
(1 row)

--hll_hashval
openGauss=# select hll_hash_integer(1) = hll_hash_integer(1);
 ?column?
----------
 t
(1 row)
```

- <> or !=

  Description: Compares the values of HLL and **hll_hashval** types to check whether they are different.

  Return type: Boolean

  Example:

```
--hll
openGauss=# select (hll_empty() || hll_hash_integer(1)) <> (hll_empty() || hll_hash_integer(2));
 ?column?
----------
 t
(1 row)

--hll_hashval
openGauss=# select hll_hash_integer(1) <> hll_hash_integer(2);
 ?column?
----------
```

```
 t
(1 row)
```

- ||

  Description: Represents the functions of **hll_add**, **hll_union**, and **hll_add_rev**.

  Return type: HLL

  Example:

```
--hll_add
openGauss=# select hll_empty() || hll_hash_integer(1);
                         ?column?
----------------------------------------------------------------------------
 \x484c4c08000002002b0900000000000000f03f3e2921ff133fbaed3e2921ff133fbaed00
(1 row)

--hll_add_rev
openGauss=# select hll_hash_integer(1) || hll_empty();
                         ?column?
----------------------------------------------------------------------------
 \x484c4c08000002002b0900000000000000f03f3e2921ff133fbaed3e2921ff133fbaed00
(1 row)

--hll_union
openGauss=# select (hll_empty() || hll_hash_integer(1)) || (hll_empty() || hll_hash_integer(2));
                                         ?column?
--------------------------------------------------------------------------------------------

\x484c4c10002000002b090000000000000000400000000000000000b3ccc49320cca1ae3e2921ff133fba
ed00
(1 row)
```

- #

  Description: Calculates the number of distinct values of an HLL. It works the same as the **hll_cardinality** function.

  Return type: int

  Example:

```
openGauss=# select #(hll_empty() || hll_hash_integer(1));
 ?column?
----------
        1
(1 row)
```

11.5.15 SEQUENCE Functions

The sequence functions provide a simple method to ensure security of multiple users for users to obtain sequence values from sequence objects.

- nextval(regclass)

  Description: Specifies an increasing sequence and returns a new value.

  NOTE

  To avoid blocking of concurrent transactions that obtain numbers from the same sequence, a nextval operation is never rolled back; that is, once a value has been fetched it is considered used, even if the transaction that did the nextval later aborts. This means that aborted transactions may leave unused "holes" in the sequence of assigned values. Therefore, GaussDB sequences cannot be used to obtain sequence without gaps.

NOTICE

The **nextval** function can be executed only on the primary node. It is not supported on standby nodes.

Return type: numeric

The **nextval** function can be invoked in either of the following ways: (In example 2, the sequence name cannot contain a dot.)

Example 1:

```
openGauss=# select nextval('seqDemo');
 nextval
---------
       2
(1 row)
```

Example 2:

```
openGauss=# select seqDemo.nextval;
 nextval
---------
       2
(1 row)
```

- currval(regclass)

  Returns the last value of **nextval** for a specified sequence in the current session. If **nextval** has not been invoked for the specified sequence in the current session, an error is reported when **currval** is invoked.

  Return type: numeric

  The **currval** function can be invoked in either of the following ways: (In example 2, the sequence name cannot contain a dot.)

  Example 1:

  ```
  openGauss=# select currval('seq1');
   currval
  ---------
         2
  (1 row)
  ```

  Example 2:

  ```
  openGauss=# select seq1.currval;
   currval
  ---------
         2
  (1 row)
  ```

- lastval()

  Description: Returns the last value of **nextval** in the current session. This function is equivalent to **currval**, but **lastval** does not have a parameter. If **nextval** has not been invoked in the current session, invoking **lastval** will report an error.

  Return type: numeric

  Example:

  ```
  openGauss=# select lastval();
   lastval
  ---------
         2
  (1 row)
  ```

- setval(regclass, numeric)

Description: Sets the current value of a sequence.

Return type: numeric

Example:

```
openGauss=# select setval('seqDemo',1);
 setval
--------
      1
(1 row)
```

- setval(regclass, numeric, Boolean)

  Description: Sets the current value of a sequence and the is_called sign.

  Return type: numeric

  Example:

```
openGauss=# select setval('seqDemo',1,true);
 setval
--------
      1
(1 row)
```

NOTE

The current session will take effect immediately after **setval** is performed. If other sessions have buffered sequence values, **setval** will take effect only after the values are used up. Therefore, to prevent sequence value conflicts, you are advised to use **setval** with caution.

Because the sequence is non-transactional, the change caused by **setval** will not be undone by transaction rollback.

NOTICE

The **nextval** function can be executed only on the primary node. It is not supported on standby nodes.

- pg_sequence_last_value(sequence_oid oid, OUT cache_value int16, OUT last_value int16)

  Description: Obtains the parameters of a specified sequence, including the cache value and current value.

  Return type: int16, int16

11.5.16 Array Functions and Operators

Array Operators

- =

  Description: Specifies whether two arrays are equal.

  Example:

```
openGauss=# SELECT ARRAY[1.1,2.1,3.1]::int[] = ARRAY[1,2,3] AS RESULT ;
 result
--------
 t
(1 row)
```

- <>

  Description: Specifies whether two arrays are not equal.

Example:

```
openGauss=# SELECT ARRAY[1,2,3] <> ARRAY[1,2,4] AS RESULT;
 result
--------
 t
(1 row)
```

- <

  Description: Specifies whether an array is less than another.

  Example:

```
openGauss=# SELECT ARRAY[1,2,3] < ARRAY[1,2,4] AS RESULT;
 result
--------
 t
(1 row)
```

- >

  Description: Specifies whether an array is greater than another.

  Example:

```
openGauss=# SELECT ARRAY[1,4,3] > ARRAY[1,2,4] AS RESULT;
 result
--------
 t
(1 row)
```

- <=

  Description: Specifies whether an array is less than another.

  Example:

```
openGauss=# SELECT ARRAY[1,2,3] <= ARRAY[1,2,3] AS RESULT;
 result
--------
 t
(1 row)
```

- >=

  Description: Specifies whether an array is greater than or equal to another.

  Example:

```
openGauss=# SELECT ARRAY[1,4,3] >= ARRAY[1,4,3] AS RESULT;
 result
--------
 t
(1 row)
```

- @>

  Description: Specifies whether an array contains another.

  Example:

```
openGauss=# SELECT ARRAY[1,4,3] @> ARRAY[3,1] AS RESULT;
 result
--------
 t
(1 row)
```

- <@

  Description: Specifies whether an array is contained in another.

  Example:

```
openGauss=# SELECT ARRAY[2,7] <@ ARRAY[1,7,4,2,6] AS RESULT;
 result
--------
 t
(1 row)
```

- &&

  Description: Specifies whether an array overlaps another (have common elements).

  Example:

  ```
  openGauss=# SELECT ARRAY[1,4,3] && ARRAY[2,1] AS RESULT;
   result
  --------
   t
  (1 row)
  ```

- ||

  Description: Array-to-array concatenation

  Example:

  ```
  openGauss=# SELECT ARRAY[1,2,3] || ARRAY[4,5,6] AS RESULT;
      result
  ---------------
   {1,2,3,4,5,6}
  (1 row)
  openGauss=# SELECT ARRAY[1,2,3] || ARRAY[[4,5,6],[7,8,9]] AS RESULT;
            result
  ---------------------------
   {{1,2,3},{4,5,6},{7,8,9}}
  (1 row)
  ```

- ||

  Description: Element-to-array concatenation

  Example:

  ```
  openGauss=# SELECT 3 || ARRAY[4,5,6] AS RESULT;
    result
  -----------
   {3,4,5,6}
  (1 row)
  ```

- ||

  Description: Array-to-element concatenation

  Example:

  ```
  openGauss=# SELECT ARRAY[4,5,6] || 7 AS RESULT;
    result
  -----------
   {4,5,6,7}
  (1 row)
  ```

Array comparisons compare the array contents element-by-element, using the default B-tree comparison function for the element data type. In multidimensional arrays, the elements are accessed in row-major order. If the contents of two arrays are equal but the dimensionality is different, the first difference in the dimensionality information determines the sort order.

Array Functions

- array_append(anyarray, anyelement)

  Description: Appends an element to the end of an array, and only supports dimension-1 arrays.

  Return type: anyarray

  Example:

  ```
  openGauss=# SELECT array_append(ARRAY[1,2], 3) AS RESULT;
   result
  ```

```
---------
 {1,2,3}
(1 row)
```

- array_prepend(anyelement, anyarray)

  Description: Appends an element to the beginning of an array, and only supports dimension-1 arrays.

  Return type: anyarray

  Example:

```
openGauss=# SELECT array_prepend(1, ARRAY[2,3]) AS RESULT;
 result
---------
 {1,2,3}
(1 row)
```

- array_cat(anyarray, anyarray)

  Description: Concatenates two arrays, and supports multi-dimensional arrays.

  Return type: anyarray

  Example:

```
openGauss=# SELECT array_cat(ARRAY[1,2,3], ARRAY[4,5]) AS RESULT;
   result
-------------
 {1,2,3,4,5}
(1 row)

openGauss=# SELECT array_cat(ARRAY[[1,2],[4,5]], ARRAY[6,7]) AS RESULT;
       result
---------------------
 {{1,2},{4,5},{6,7}}
(1 row)
```

- array_union(anyarray, anyarray)

  Description: Concatenates two arrays, and supports only one-dimensional arrays.

  Return type: anyarray

  Example:

```
openGauss=# SELECT array_union(ARRAY[1,2,3], ARRAY[3,4,5]) AS RESULT;
   result
-------------
 {1,2,3,3,4,5}
(1 row)
```

- array_union_distinct(anyarray, anyarray)

  Description: Concatenates two arrays and deduplicates them. Only one-dimensional arrays are supported.

  Return type: anyarray

  Example:

```
openGauss=# SELECT array_union_distinct(ARRAY[1,2,3], ARRAY[3,4,5]) AS RESULT;
   result
-------------
 {1,2,3,4,5}
(1 row)
```

- array_intersect(anyarray, anyarray)

  Description: Intersects two arrays. Only one-dimensional arrays are supported.

  Return type: anyarray

  Example:

```
openGauss=# SELECT array_intersect(ARRAY[1,2,3], ARRAY[3,4,5]) AS RESULT;
   result
-------------
 {3}
(1 row)
```

- array_intersect_distinct(anyarray, anyarray)

  Description: Intersects two arrays and deduplicates them. Only one-dimensional arrays are supported.

  Return type: anyarray

  Example:

```
openGauss=# SELECT array_intersect_distinct(ARRAY[1,2,2], ARRAY[2,2,4,5]) AS RESULT;
   result
-------------
 {2}
(1 row)
```

- array_except(anyarray, anyarray)

  Description: Calculates the difference between two arrays. Only one-dimensional arrays are supported.

  Return type: anyarray

  Example:

```
openGauss=# SELECT array_except(ARRAY[1,2,3], ARRAY[3,4,5]) AS RESULT;
   result
-------------
 {1,2}
(1 row)
```

- array_except_distinct(anyarray, anyarray)

  Description: Calculates the difference between two arrays and deduplicates them. Only one-dimensional arrays are supported.

  Return type: anyarray

  Example:

```
openGauss=# SELECT array_except_distinct(ARRAY[1,2,2,3], ARRAY[3,4,5]) AS RESULT;
   result
-------------
 {1,2}
(1 row)
```

- array_ndims(anyarray)

  Description: Returns the number of dimensions of an array.

  Return type: int

  Example:

```
openGauss=# SELECT array_ndims(ARRAY[[1,2,3], [4,5,6]]) AS RESULT;
 result
--------
      2
(1 row)
```

- array_dims(anyarray)

  Description: Returns the low-order flag bits and high-order flag bits of each dimension in an array.

  Return type: text

  Example:

```
openGauss=# SELECT array_dims(ARRAY[[1,2,3], [4,5,6]]) AS RESULT;
   result
------------
```

```
 [1:2][1:3]
(1 row)
```

- array_length(anyarray, int)

  Description: Returns the length of the requested array dimension. **int** is the requested array dimension.

  Return type: int

  Example:

```
openGauss=# SELECT array_length(array[1,2,3], 1) AS RESULT;
 result
--------
      3
(1 row)

openGauss=# SELECT array_length(array[[1,2,3],[4,5,6]], 2) AS RESULT;
 result
--------
      3
(1 row)
```

- array_lower(anyarray, int)

  Description: Returns lower bound of the requested array dimension. **int** is the requested array dimension.

  Return type: int

  Example:

```
openGauss=# SELECT array_lower('[0:2]={1,2,3}'::int[], 1) AS RESULT;
 result
--------
      0
(1 row)
```

- array_upper(anyarray, int)

  Description: Returns upper bound of the requested array dimension. **int** is the requested array dimension.

  Return type: int

  Example:

```
openGauss=# SELECT array_upper(ARRAY[1,8,3,7], 1) AS RESULT;
 result
--------
      4
(1 row)
```

- array_upper(anyarray, int)

  Description: Returns upper bound of the requested array dimension. **int** is the requested array dimension.

  Return type: int

  Example:

```
openGauss=# SELECT array_upper(ARRAY[1,8,3,7], 1) AS RESULT;
 result
--------
      4
(1 row)
```

- array_remove(anyarray, anyelement)

  Description: Removes all specified elements from an array. Only one-dimensional arrays are supported.

  Return type: anyarray

  Example:

```
openGauss=# SELECT array_remove(ARRAY[1,8,8,7], 8) AS RESULT;
 result
--------
 {1,7}
(1 row)
```

- array_to_string(anyarray, text [, text])

  Description: Uses the first **text** as the new delimiter and the second **text** to replace **NULL** values.

  Return type: text

  Example:

```
openGauss=# SELECT array_to_string(ARRAY[1, 2, 3, NULL, 5], ',', '*') AS RESULT;
  result
-----------
 1,2,3,*,5
(1 row)
```

- array_delete(anyarray)

  Description: Clears elements in an array and returns an empty array of the same type.

  Return type: anyarray

  Example:

```
openGauss=# SELECT array_delete(ARRAY[1,8,3,7]) AS RESULT;
 result
--------
 {}
(1 row)
```

- array_deleteidx(anyarray, int)

  Description: Deletes specified subscript elements from an array and returns an array consisting of the remaining elements.

  Return type: anyarray

  Example:

```
openGauss=# SELECT array_deleteidx(ARRAY[1,2,3,4,5], 1) AS RESULT;
  result
-----------
 {2,3,4,5}
(1 row)
```

- array_extendnull(anyarray, int)

  Description: Adds a specified number of null elements to the end of an array.

  Return type: anyarray

  Example:

```
openGauss=# SELECT array_extendnull(ARRAY[1,8,3,7],1) AS RESULT;
    result
--------------
{1,8,3,7,null}
(1 row)
```

- array_trim(anyarray, int)

  Description: Deletes a specified number of elements from the end of an array.

  Return type: anyarray

  Example:

```
openGauss=# SELECT array_trim(ARRAY[1,8,3,7],1) AS RESULT;
 result
---------
 {1,8,3}
(1 row)
```

- array_exists(anyarray, int)

  Description: Checks whether the second parameter is a valid subscript of an array.

  Return type: Boolean

  Example:

```
openGauss=# SELECT array_exists(ARRAY[1,8,3,7],1) AS RESULT;
 result
--------
 t
(1 row)
```

- array_next(anyarray, int)

  Description: Returns the subscript of the element following a specified subscript in an array based on the second input parameter.

  Return type: int

  Example:

```
openGauss=# SELECT array_next(ARRAY[1,8,3,7],1) AS RESULT;
 result
--------
      2
(1 row)
```

- array_prior(anyarray, int)

  Description: Returns the subscript of the element followed by a specified subscript in an array based on the second input parameter.

  Return type: int

  Example:

```
openGauss=# SELECT array_prior(ARRAY[1,8,3,7],2) AS RESULT;
 result
--------
  1
(1 row)
```

- string_to_array(text, text [, text])

  Description: Uses the second **text** as the new delimiter and the third **text** as the substring to be replaced by **NULL** values. A substring can be replaced by **NULL** values only when it is the same as the third **text**.

  Return type: text[]

  Example:

```
openGauss=# SELECT string_to_array('xx~^~yy~^~zz', '~^~', 'yy') AS RESULT;
    result
--------------
 {xx,NULL,zz}
(1 row)
openGauss=# SELECT string_to_array('xx~^~yy~^~zz', '~^~', 'y') AS RESULT;
   result
------------
 {xx,yy,zz}
(1 row)
```

- unnest(anyarray)

  Description: Expands an array to a set of rows.

  Return type: setof anyelement

  Example:

```
openGauss=# SELECT unnest(ARRAY[1,2]) AS RESULT;
 result
```

```
--------
      1
      2
(2 rows)
```

In **string_to_array**, if the delimiter parameter is NULL, each character in the input string will become a separate element in the resulting array. If the delimiter is an empty string, then the entire input string is returned as a one-element array. Otherwise the input string is split at each occurrence of the delimiter string.

In **string_to_array**, if the null-string parameter is omitted or NULL, none of the substrings of the input will be replaced by NULL.

In **array_to_string**, if the null-string parameter is omitted or NULL, any null elements in the array are simply skipped and not represented in the output string.

- _pg_keysequal

  Description: Checks whether two smallint arrays are the same.

  Parameter: smallint[], smallint[]

  Return type: Boolean

11.5.17 Range Functions and Operators

Range Operators

- =

  Description: Equals

  Example:

  ```
  openGauss=# SELECT int4range(1,5) = '[1,4]'::int4range AS RESULT;
   result
  --------
   t
  (1 row)
  ```

- <>

  Description: Does not equal to

  Example:

  ```
  openGauss=# SELECT numrange(1.1,2.2) <> numrange(1.1,2.3) AS RESULT;
   result
  --------
   t
  (1 row)
  ```

- <

  Description: Is less than

  Example:

  ```
  openGauss=# SELECT int4range(1,10) < int4range(2,3) AS RESULT;
   result
  --------
   t
  (1 row)
  ```

- >

  Description: Is greater than

  Example:

  ```
  openGauss=# SELECT int4range(1,10) > int4range(1,5) AS RESULT;
   result
  ```

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

- <=

  Description: Is less than or equals

  Example:

```
openGauss=# SELECT numrange(1.1,2.2) <= numrange(1.1,2.2) AS RESULT;
 result
--------
 t
(1 row)
```

- >=

  Description: Is greater than or equals

  Example:

```
openGauss=# SELECT numrange(1.1,2.2) >= numrange(1.1,2.0) AS RESULT;
 result
--------
 t
(1 row)
```

- @>

  Description: Contains ranges

  Example:

```
openGauss=# SELECT int4range(2,4) @> int4range(2,3) AS RESULT;
 result
--------
 t
(1 row)
```

- @>

  Description: Contains elements

  Example:

```
openGauss=# SELECT '[2011-01-01,2011-03-01)'::tsrange @> '2011-01-10'::timestamp AS RESULT;
 result
--------
 t
(1 row)
```

- <@

  Description: Range is contained by

  Example:

```
openGauss=# SELECT int4range(2,4) <@ int4range(1,7) AS RESULT;
 result
--------
 t
(1 row)
```

- <@

  Description: Element is contained by

  Example:

```
openGauss=# SELECT 42 <@ int4range(1,7) AS RESULT;
 result
--------
 f
(1 row)
```

- &&

  Description: Overlap (have points in common)

Example:

```
openGauss=# SELECT int8range(3,7) && int8range(4,12) AS RESULT;
 result
--------
 t
(1 row)
```

- <<

  Description: Strictly left of

  Example:

```
openGauss=# SELECT int8range(1,10) << int8range(100,110) AS RESULT;
 result
--------
 t
(1 row)
```

- >>

  Description: Strictly right of

  Example:

```
openGauss=# SELECT int8range(50,60) >> int8range(20,30) AS RESULT;
 result
--------
 t
(1 row)
```

- &<

  Description: Does not extend to the right of

  Example:

```
openGauss=# SELECT int8range(1,20) &< int8range(18,20) AS RESULT;
 result
--------
 t
(1 row)
```

- &>

  Description: Does not extend to the left of

  Example:

```
openGauss=# SELECT int8range(7,20) &> int8range(5,10) AS RESULT;
 result
--------
 t
(1 row)
```

- -|-

  Description: Is adjacent to

  Example:

```
openGauss=# SELECT numrange(1.1,2.2) -|- numrange(2.2,3.3) AS RESULT;
 result
--------
 t
(1 row)
```

- +

  Description: Union

  Example:

```
openGauss=# SELECT numrange(5,15) + numrange(10,20) AS RESULT;
 result
--------
 [5,20)
(1 row)
```

- *

  Description: Intersection

  Example:

  ```
  openGauss=# SELECT int8range(5,15) * int8range(10,20) AS RESULT;
   result
  ---------
   [10,15)
  (1 row)
  ```

- -

  Description: Difference

  Example:

  ```
  openGauss=# SELECT int8range(5,15) - int8range(10,20) AS RESULT;
   result
  --------
   [5,10)
  (1 row)
  ```

The simple comparison operators **<**, **>**, **<=**, and **>=** compare the lower bounds first, and only if those are equal, compare the upper bounds.

The **<<**, **>>**, and **-|-** operators always return false when an empty range is involved; that is, an empty range is not considered to be either before or after any other range.

The union and difference operators will fail if the resulting range would need to contain two disjoint sub-ranges.

Range Functions

- numrange(numeric, numeric, [text])

  Description: Specifies a range.

  Return type: Range's element type

  Example:

  ```
  openGauss=# SELECT numrange(1.1,2.2) AS RESULT;
   result
  --------
  [1.1,2.2)
  (1 row)
  openGauss=# SELECT numrange(1.1,2.2, '()') AS RESULT;
   result
  --------
  (1.1,2.2)
  (1 row)
  ```

- lower(anyrange)

  Description: Lower bound of a range

  Return type: Range's element type

  Example:

  ```
  openGauss=# SELECT lower(numrange(1.1,2.2)) AS RESULT;
   result
  --------
     1.1
  (1 row)
  ```

- upper(anyrange)

  Description: Upper bound of a range

  Return type: Range's element type

Example:

```
openGauss=# SELECT upper(numrange(1.1,2.2)) AS RESULT;
 result
--------
    2.2
(1 row)
```

- isempty(anyrange)

  Description: Is the range empty?

  Return type: Boolean

  Example:

```
openGauss=# SELECT isempty(numrange(1.1,2.2)) AS RESULT;
 result
--------
 f
(1 row)
```

- lower_inc(anyrange)

  Description: Is the lower bound inclusive?

  Return type: Boolean

  Example:

```
openGauss=# SELECT lower_inc(numrange(1.1,2.2)) AS RESULT;
 result
--------
 t
(1 row)
```

- upper_inc(anyrange)

  Description: Is the upper bound inclusive?

  Return type: Boolean

  Example:

```
openGauss=# SELECT upper_inc(numrange(1.1,2.2)) AS RESULT;
 result
--------
 f
(1 row)
```

- lower_inf(anyrange)

  Description: Is the lower bound infinite?

  Return type: Boolean

  Example:

```
openGauss=# SELECT lower_inf('(,)'::daterange) AS RESULT;
 result
--------
 t
(1 row)
```

- upper_inf(anyrange)

  Description: Is the upper bound infinite?

  Return type: Boolean

  Example:

```
openGauss=# SELECT upper_inf('(,)'::daterange) AS RESULT;
 result
--------
 t
(1 row)
```

The **lower** and **upper** functions return null if the range is empty or the requested bound is infinite. The **lower_inc**, **upper_inc**, **lower_inf**, and **upper_inf** functions all return false for an empty range.

- elem_contained_by_range(anyelement, anyrange)

  Description: Determines whether an element is within the range.

  Return type: Boolean

  Example:

```
openGauss=# SELECT elem_contained_by_range('2', numrange(1.1,2.2));
 elem_contained_by_range
-------------------------
 t
(1 row)
```

11.5.18 Aggregate Functions

Aggregate Functions

- sum(expression)

  Description: Specifies the sum of expressions across all input values.

  Return type:

  Generally, same as the argument data type. In the following cases, type conversion occurs:

  – **BIGINT** for **SMALLINT** or **INT** arguments
  – **NUMBER** for **BIGINT** arguments
  – **DOUBLE PRECISION** for floating-point arguments

  Example:

```
openGauss=# SELECT SUM(ss_ext_tax) FROM tpcds.STORE_SALES;
  sum
--------------
 213267594.69
(1 row)
```

- max(expression)

  Description: Specifies the maximum value of expression across all input values.

  Parameter type: any array, numeric, string, or date/time type

  Return type: same as the argument type

  Example:

```
openGauss=# SELECT MAX(inv_quantity_on_hand) FROM tpcds.inventory;
```

- min(expression)

  Description: Specifies the minimum value of expression across all input values.

  Parameter type: any array, numeric, string, or date/time type

  Return type: same as the argument type

  Example:

```
openGauss=# SELECT MIN(inv_quantity_on_hand) FROM tpcds.inventory;
 min
-----
   0
(1 row)
```

- avg(expression)

  Description: Specifies the average (arithmetic mean) of all input values.

  Return type:

  NUMBER for any integer-type argument.

  DOUBLE PRECISION for floating-point arguments.

  otherwise the same as the argument data type.

  Example:

```
openGauss=# SELECT AVG(inv_quantity_on_hand) FROM tpcds.inventory;
       avg
----------------------
 500.0387129084044604
(1 row)
```

- count(expression)

  Description: Specifies the number of input rows for which the value of the expression is not null.

  Return type: bigint

  Example:

```
openGauss=# SELECT COUNT(inv_quantity_on_hand) FROM tpcds.inventory;
  count
----------
 11158087
(1 row)
```

- count(*)

  Description: Returns the number of input rows.

  Return type: bigint

  Example:

```
openGauss=# SELECT COUNT(*) FROM tpcds.inventory;
  count
----------
 11745000
(1 row)
```

- median(expression) [over (query partition clause)]

  Description: Returns the median of an expression. **NULL** will be ignored by the median function during calculation. The **DISTINCT** keyword can be used to exclude duplicate records in an expression. The data type of the input expression can be numeric (including integer, double, and bigint) or interval. For other data types, the median cannot be calculated.

  Return type: double or interval

  Example:

```
select median(id) from (values(1), (2), (3), (4), (null)) test(id);
 median
--------
    2.5
(1 row)
```

- array_agg(expression)

  Description: Concatenates input values, including nulls, into an array.

  Return type: array of the argument type

  Example:

```
openGauss=# SELECT ARRAY_AGG(sr_fee) FROM tpcds.store_returns WHERE sr_customer_sk = 2;
   array_agg
```

```
---------------
 {22.18,63.21}
(1 row)
```

- string_agg(expression, delimiter)

  Description: Concatenates input values into a string, separated by delimiter.

  Return type: same as the argument type

  Example:

```
openGauss=# SELECT string_agg(sr_item_sk, ',') FROM tpcds.store_returns where sr_item_sk < 3;
         string_agg
---------------------------------------------------------------------------------
------------------------------
 1,2,1,2,2,1,1,2,2,1,2,1,2,1,1,1,2,1,1,1,1,1,2,1,1,1,1,1,2,2,1,1,1,1,1,1,1,1,1,2,
2,1,1,1,1,1,1,2,2,1,1,2,1,1,1
(1 row)
```

- listagg(expression [, delimiter]) WITHIN GROUP(ORDER BY order-list)

  Description: Sorts aggregation column data according to the mode specified by **WITHIN GROUP** and concatenates the data to a string using the specified delimiter.

  – **expression**: Mandatory. It specifies an aggregation column name or a column-based valid expression. It does not support the **DISTINCT** keyword and the **VARIADIC** parameter.
  – **delimiter**: Optional. It specifies a delimiter, which can be a string constant or a deterministic expression based on a group of columns. The default value is empty.
  – **order-list**: Mandatory. It specifies the sorting mode in a group.

  Return type: text

  Example:

  The aggregation column is of the text character set type.

```
openGauss=# SELECT deptno, listagg(ename, ',') WITHIN GROUP(ORDER BY ename) AS employees 
FROM emp GROUP BY deptno;
 deptno |              employees
--------+--------------------------------------
     10 | CLARK,KING,MILLER
     20 | ADAMS,FORD,JONES,SCOTT,SMITH
     30 | ALLEN,BLAKE,JAMES,MARTIN,TURNER,WARD
(3 rows)
```

  The aggregation column is of the integer type.

```
openGauss=# SELECT deptno, listagg(mgrno, ',') WITHIN GROUP(ORDER BY mgrno NULLS FIRST) AS 
mgrnos FROM emp GROUP BY deptno;
 deptno |            mgrnos
--------+-------------------------------
     10 | 7782,7839
     20 | 7566,7566,7788,7839,7902
     30 | 7698,7698,7698,7698,7698,7839
(3 rows)
```

  The aggregation column is of the floating point type.

```
openGauss=# SELECT job, listagg(bonus, '($); ') WITHIN GROUP(ORDER BY bonus DESC) || '($)' AS 
bonus FROM emp GROUP BY job;
    job     |                      bonus
------------+-------------------------------------------------
 CLERK      | 10234.21($); 2000.80($); 1100.00($); 1000.22($)
 PRESIDENT  | 23011.88($)
 ANALYST    | 2002.12($); 1001.01($)
 MANAGER    | 10000.01($); 2399.50($); 999.10($)
 SALESMAN   | 1000.01($); 899.00($); 99.99($); 9.00($)
(5 rows)
```

The aggregation column is of the time type.

```
openGauss=# SELECT deptno, listagg(hiredate, ', ') WITHIN GROUP(ORDER BY hiredate DESC) AS
hiredates FROM emp GROUP BY deptno;
 deptno |                                                    hiredates
--------
+-----------------------------------------------------------------------------------------------------------------------
---------
     10 | 1982-01-23 00:00:00, 1981-11-17 00:00:00, 1981-06-09 00:00:00
     20 | 2001-04-02 00:00:00, 1999-12-17 00:00:00, 1987-05-23 00:00:00, 1987-04-19 00:00:00,
1981-12-03 00:00:00
     30 | 2015-02-20 00:00:00, 2010-02-22 00:00:00, 1997-09-28 00:00:00, 1981-12-03 00:00:00,
1981-09-08 00:00:00, 1981-05-01 00:00:00
(3 rows)
```

The aggregation column is of the time interval type.

```
openGauss=# SELECT deptno, listagg(vacationTime, '; ') WITHIN GROUP(ORDER BY vacationTime
DESC) AS vacationTime FROM emp GROUP BY deptno;
 deptno |                                    vacationtime
--------+-------------------------------------------------------------------------------------
     10 | 1 year 30 days; 40 days; 10 days
     20 | 70 days; 36 days; 9 days; 5 days
     30 | 1 year 1 mon; 2 mons 10 days; 30 days; 12 days 12:00:00; 4 days 06:00:00; 24:00:00
(3 rows)
```

By default, the delimiter is empty.

```
openGauss=# SELECT deptno, listagg(job) WITHIN GROUP(ORDER BY job) AS jobs FROM emp
GROUP BY deptno;
 deptno |                    jobs
--------+----------------------------------------------
     10 | CLERKMANAGERPRESIDENT
     20 | ANALYSTANALYSTCLERKCLERKMANAGER
     30 | CLERKMANAGERSALESMANSALESMANSALESMANSALESMAN
(3 rows)
```

When **listagg** is used as a window function, the **OVER** clause does not support the window sorting of **ORDER BY**, and the **listagg** column is an ordered aggregation of the corresponding groups.

```
openGauss=# SELECT deptno, mgrno, bonus, listagg(ename,'; ') WITHIN GROUP(ORDER BY hiredate)
OVER(PARTITION BY deptno) AS employees FROM emp;
 deptno | mgrno |  bonus   |                 employees
--------+-------+----------+-------------------------------------------
     10 |  7839 | 10000.01 | CLARK; KING; MILLER
     10 |       | 23011.88 | CLARK; KING; MILLER
     10 |  7782 | 10234.21 | CLARK; KING; MILLER
     20 |  7566 |  2002.12 | FORD; SCOTT; ADAMS; SMITH; JONES
     20 |  7566 |  1001.01 | FORD; SCOTT; ADAMS; SMITH; JONES
     20 |  7788 |  1100.00 | FORD; SCOTT; ADAMS; SMITH; JONES
     20 |  7902 |  2000.80 | FORD; SCOTT; ADAMS; SMITH; JONES
     20 |  7839 |   999.10 | FORD; SCOTT; ADAMS; SMITH; JONES
     30 |  7839 |  2399.50 | BLAKE; TURNER; JAMES; MARTIN; WARD; ALLEN
     30 |  7698 |     9.00 | BLAKE; TURNER; JAMES; MARTIN; WARD; ALLEN
     30 |  7698 |  1000.22 | BLAKE; TURNER; JAMES; MARTIN; WARD; ALLEN
     30 |  7698 |    99.99 | BLAKE; TURNER; JAMES; MARTIN; WARD; ALLEN
     30 |  7698 |  1000.01 | BLAKE; TURNER; JAMES; MARTIN; WARD; ALLEN
     30 |  7698 |   899.00 | BLAKE; TURNER; JAMES; MARTIN; WARD; ALLEN
(14 rows)
```

- covar_pop(Y, X)

  Description: Specifies the overall covariance.

  Return type: double precision

  Example:

```
openGauss=# SELECT COVAR_POP(sr_fee, sr_net_loss) FROM tpcds.store_returns WHERE
sr_customer_sk < 1000;
    covar_pop
------------------
```

```
 829.749627587403
(1 row)
```

- covar_samp(Y, X)

  Description: Specifies the sample covariance.

  Return type: double precision

  Example:

```
openGauss=# SELECT COVAR_SAMP(sr_fee, sr_net_loss) FROM tpcds.store_returns WHERE
sr_customer_sk < 1000;
    covar_samp
------------------
 830.052235037289
(1 row)
```

- stddev_pop(expression)

  Description: Specifies the overall standard deviation.

  Return type: **double precision** for floating-point arguments, otherwise **numeric**

  Example:

```
openGauss=# SELECT STDDEV_POP(inv_quantity_on_hand) FROM tpcds.inventory WHERE
inv_warehouse_sk = 1;
    stddev_pop
------------------
 289.224294957556
(1 row)
```

- stddev_samp(expression)

  Description: Specifies the sample standard deviation of the input values.

  Return type: **double precision** for floating-point arguments, otherwise **numeric**

  Example:

```
openGauss=# SELECT STDDEV_SAMP(inv_quantity_on_hand) FROM tpcds.inventory WHERE
inv_warehouse_sk = 1;
   stddev_samp
------------------
 289.224359757315
(1 row)
```

- var_pop(expression)

  Description: Specifies the population variance of the input values (square of the population standard deviation).

  Return type: **double precision** for floating-point arguments, otherwise **numeric**

  Example:

```
openGauss=# SELECT VAR_POP(inv_quantity_on_hand) FROM tpcds.inventory WHERE
inv_warehouse_sk = 1;
      var_pop
--------------------
 83650.692793695475
(1 row)
```

- var_samp(expression)

  Description: Specifies the sample variance of the input values (square of the sample standard deviation).

  Return type: **double precision** for floating-point arguments, otherwise **numeric**

  Example:

```
openGauss=# SELECT VAR_SAMP(inv_quantity_on_hand) FROM tpcds.inventory WHERE
inv_warehouse_sk = 1;
      var_samp
--------------------
 83650.730277028768
(1 row)
```

- bit_and(expression)

  Description: bitwise AND of all non-null input values, or null if none

  Return type: same as the argument type

  Example:

```
openGauss=# SELECT BIT_AND(inv_quantity_on_hand) FROM tpcds.inventory WHERE
inv_warehouse_sk = 1;
 bit_and
---------
       0
(1 row)
```

- bit_or(expression)

  Description: bitwise OR of all non-null input values, or null if none

  Return type: same as the argument type

  Example:

```
openGauss=# SELECT BIT_OR(inv_quantity_on_hand) FROM tpcds.inventory WHERE
inv_warehouse_sk = 1;
 bit_or
--------
   1023
(1 row)
```

- bool_and(expression)

  Description: Its value is **true** if all input values are **true**, otherwise **false**.

  Return type: Boolean

  Example:

```
openGauss=# SELECT bool_and(100 <2500);
 bool_and
----------
 t
(1 row)
```

- bool_or(expression)

  Description: Its value is **true** if at least one input value is **true**, otherwise **false**.

  Return type: Boolean

  Example:

```
openGauss=# SELECT bool_or(100 <2500);
 bool_or
----------
 t
(1 row)
```

- corr(Y, X)

  Description: Specifies the correlation coefficient.

  Return type: double precision

  Example:

```
openGauss=# SELECT CORR(sr_fee, sr_net_loss) FROM tpcds.store_returns WHERE sr_customer_sk <
1000;
       corr
-------------------
```

```
 .0381383624904186
(1 row)
```

- every(expression)

  Description: Equivalent to **bool_and**

  Return type: Boolean

  Example:

```
openGauss=# SELECT every(100 <2500);
 every
-------
 t
(1 row)
```

- regr_avgx(Y, X)

  Description: Specifies the average of the independent variable (**sum(X)/N**).

  Return type: double precision

  Example:

```
openGauss=# SELECT REGR_AVGX(sr_fee, sr_net_loss) FROM tpcds.store_returns WHERE 
sr_customer_sk < 1000;
    regr_avgx
------------------
 578.606576740795
(1 row)
```

- regr_avgy(Y, X)

  Description: Specifies the average of the dependent variable (**sum(Y)/N**).

  Return type: double precision

  Example:

```
openGauss=# SELECT REGR_AVGY(sr_fee, sr_net_loss) FROM tpcds.store_returns WHERE 
sr_customer_sk < 1000;
    regr_avgy
------------------
 50.0136711629602
(1 row)
```

- regr_count(Y, X)

  Description: Specifies the number of input rows in which both expressions are non-null.

  Return type: bigint

  Example:

```
openGauss=# SELECT REGR_COUNT(sr_fee, sr_net_loss) FROM tpcds.store_returns WHERE 
sr_customer_sk < 1000;
 regr_count
------------
       2743
(1 row)
```

- regr_intercept(Y, X)

  Description: Specifies the y-intercept of the least-squares-fit linear equation determined by the (X, Y) pairs.

  Return type: double precision

  Example:

```
openGauss=# SELECT REGR_INTERCEPT(sr_fee, sr_net_loss) FROM tpcds.store_returns WHERE 
sr_customer_sk < 1000;
  regr_intercept
------------------
 49.2040847848607
(1 row)
```

- regr_r2(Y, X)

  Description: Specifies the square of the correlation coefficient.

  Return type: double precision

  Example:

```
openGauss=# SELECT REGR_R2(sr_fee, sr_net_loss) FROM tpcds.store_returns WHERE sr_customer_sk
< 1000;
      regr_r2
--------------------
 .00145453469345058
(1 row)
```

- regr_slope(Y, X)

  Description: Specifies the slope of the least-squares-fit linear equation determined by the (X, Y) pairs.

  Return type: double precision

  Example:

```
openGauss=# SELECT REGR_SLOPE(sr_fee, sr_net_loss) FROM tpcds.store_returns WHERE
sr_customer_sk < 1000;
     regr_slope
--------------------
 .00139920009665259
(1 row)
```

- regr_sxx(Y, X)

  Description: **sum(X^2) - sum(X)^2/N** (sum of squares of the independent variables)

  Return type: double precision

  Example:

```
openGauss=# SELECT REGR_SXX(sr_fee, sr_net_loss) FROM tpcds.store_returns WHERE sr_customer_sk
< 1000;
     regr_sxx
------------------
 1626645991.46135
(1 row)
```

- regr_sxy(Y, X)

  Description: **sum(X*Y) - sum(X) * sum(Y)/N** ("sum of products" of independent times dependent variable)

  Return type: double precision

  Example:

```
openGauss=# SELECT REGR_SXY(sr_fee, sr_net_loss) FROM tpcds.store_returns WHERE sr_customer_sk
< 1000;
     regr_sxy
------------------
 2276003.22847225
(1 row)
```

- regr_syy(Y, X)

  Description: **sum(Y^2) - sum(Y)^2/N** ("sum of squares" of the dependent variable)

  Return type: double precision

  Example:

```
openGauss=# SELECT REGR_SYY(sr_fee, sr_net_loss) FROM tpcds.store_returns WHERE sr_customer_sk
< 1000;
    regr_syy
-----------------
```

```
 2189417.6547314
(1 row)
```

- stddev(expression)

  Description: Specifies the alias of **stddev_samp**.

  Return type: **double precision** for floating-point arguments, otherwise **numeric**

  Example:

```
openGauss=# SELECT STDDEV(inv_quantity_on_hand) FROM tpcds.inventory WHERE
inv_warehouse_sk = 1;
     stddev
------------------
 289.224359757315
(1 row)
```

- variance(expexpression,ression)

  Description: Specifies the alias of **var_samp**.

  Return type: **double precision** for floating-point arguments, otherwise **numeric**

  Example:

```
openGauss=# SELECT VARIANCE(inv_quantity_on_hand) FROM tpcds.inventory WHERE
inv_warehouse_sk = 1;
     variance
--------------------
 83650.730277028768
(1 row)
```

- delta

  Description: Returns the difference between the current row and the previous row.

  Parameter: numeric

  Return type: numeric

- checksum(expression)

  Description: Returns the **CHECKSUM** value of all input values. This function can be used to check whether the data in the tables is the same before and after the backup, restoration, or migration of GaussDB (databases other than GaussDB are not supported). Before and after database backup, database restoration, or data migration, you need to manually run SQL commands to obtain the execution results. Compare the obtained execution results to check whether the data in the tables before and after the backup or migration is the same.

  NOTE

  - For large tables, the execution of the **CHECKSUM** function may take a long time.
  - If the **CHECKSUM** values of two tables are different, it indicates that the contents of the two tables are different. Using the hash function in the **CHECKSUM** function may incur conflicts. There is low possibility that two tables with different contents may have the same **CHECKSUM** value. The same problem may occur when **CHECKSUM** is used for columns.
  - If the time type is timestamp, timestamptz, or smalldatetime, ensure that the time zone settings are the same when calculating the **CHECKSUM** value.

  – If the **CHECKSUM** value of a column is calculated and the column type can be changed to TEXT by default, set *expression* to the column name.

- If the **CHECKSUM** value of a column is calculated and the column type cannot be converted to TEXT by default, set *expression* to *Column name*::**TEXT**.
- If the **CHECKSUM** value of all columns is calculated, set *expression* to *Table name*::**TEXT**.

The following types of data can be converted into TEXT types by default: char, name, int8, int2, int1, int4, raw, pg_node_tree, float4, float8, bpchar, varchar, nvarchar, nvarchar2, date, timestamp, timestamptz, numeric, and smalldatetime. Other types need to be forcibly converted to TEXT.

Return type: numeric

Example:

The following shows the **CHECKSUM** value of a column that can be converted to the TEXT type by default:

```
openGauss=# SELECT CHECKSUM(inv_quantity_on_hand) FROM tpcds.inventory;
     checksum
-------------------
 24417258945265247
(1 row)
```

The following shows the **CHECKSUM** value of a column that cannot be converted to the TEXT type by default. Note that the **CHECKSUM** parameter is set to *Column name*::**TEXT**.

```
openGauss=# SELECT CHECKSUM(inv_quantity_on_hand::TEXT) FROM tpcds.inventory;
     checksum
-------------------
 24417258945265247
(1 row)
```

The following shows the **CHECKSUM** value of all columns in a table. Note that the **CHECKSUM** parameter is set to *Table name*::**TEXT**. The table name is not modified by its schema.

```
openGauss=# SELECT CHECKSUM(inventory::TEXT) FROM tpcds.inventory;
     checksum
-------------------
 25223696246875800
(1 row)
```

- first(anyelement)

  Description: Returns the first non-null input.

  Return type: anyelement

```
openGauss=# select * from tba;
name
-----
A
A
D
(4 rows)

openGauss=# select first(name) from tba;
first
-----
A
(1 rows)
```

- last(anyelement)

  Description: Returns the last non-null input.

  Return type: anyelement

```
openGauss=# select * from tba;
name
```

```
-----
A
A
D
(4 rows)

openGauss=# select last(name) from tba;
last
-----
D
(1 rows)
```

- mode() within group (order by value anyelement)

  Description: Returns the value with the highest occurrence frequency in a column. If multiple values have the same frequency, the smallest value is returned. The sorting mode is the same as the default sorting mode of the column type. **value** is an input parameter and can be of any type.

  Return type: same as the input parameter type

  Example:

```
openGauss=# select mode() within group (order by value) from (values(1, 'a'), (2, 'b'), (2, 'c'))
v(value, tag);
 mode
------
    2
(1 row)
openGauss=# select mode() within group (order by tag) from (values(1, 'a'), (2, 'b'), (2, 'c')) v(value,
tag);
 mode
------
 a
(1 row)
```

11.5.19 Window Functions

Window Functions

Currently, column-store tables only support **rank(expression)** and **row_number(expression)** functions.

Window functions and the **OVER** clause are used together. The **OVER** clause is used for grouping data and sorting the elements in a group. Window functions are used for generating sequence numbers for the values in the group.

NOTE

order by in a window function must be followed by a column name. If it is followed by a number, the number is processed as a constant value and the target column is not ranked.

- RANK()

  Description: Generates non-consecutive sequence numbers for the values in each group. The same values have the same sequence number.

  Return type: bigint

  Example:

```
openGauss=# SELECT d_moy, d_fy_week_seq, rank() OVER(PARTITION BY d_moy ORDER BY
d_fy_week_seq) FROM tpcds.date_dim WHERE d_moy < 4 AND d_fy_week_seq < 7 ORDER BY 1,2;
   d_moy | d_fy_week_seq | rank
-------+---------------+------
     1 |             1 |    1
     1 |             1 |    1
     1 |             1 |    1
     1 |            1 |    1
     1 |            1 |    1
     1 |            1 |    1
     1 |            1 |    1
     1 |            2 |    8
     1 |            2 |    8
     1 |            2 |    8
     1 |            2 |    8
     1 |            2 |    8
     1 |            2 |    8
     1 |            2 |    8
     1 |            3 |   15
     1 |            3 |   15
     1 |            3 |   15
     1 |            3 |   15
     1 |            3 |   15
     1 |            3 |   15
     1 |            3 |   15
     1 |            4 |   22
     1 |            4 |   22
     1 |            4 |   22
     1 |            4 |   22
     1 |            4 |   22
     1 |            4 |   22
     1 |            4 |   22
     1 |            5 |   29
     1 |            5 |   29
     2 |            5 |    1
     2 |            5 |    1
     2 |            5 |    1
     2 |            5 |    1
     2 |            5 |    1
     2 |            6 |    6
     2 |            6 |    6
     2 |            6 |    6
     2 |            6 |    6
     2 |            6 |    6
     2 |            6 |    6
     2 |            6 |    6
(42 rows)
```

- ROW_NUMBER()

  Description: Generates consecutive sequence numbers for the values in each group. The same values have different sequence numbers.

  Return type: bigint

  Example:

```
openGauss=# SELECT d_moy, d_fy_week_seq, Row_number() OVER(PARTITION BY d_moy ORDER BY d_fy_week_seq) FROM tpcds.date_dim  WHERE d_moy < 4 AND d_fy_week_seq < 7 ORDER BY 1,2;
 d_moy | d_fy_week_seq | row_number
-------+---------------+------------
     1 |            1 |          1
     1 |            1 |          2
     1 |            1 |          3
     1 |            1 |          4
     1 |            1 |          5
     1 |            1 |          6
     1 |            1 |          7
     1 |            2 |          8
     1 |            2 |          9
     1 |            2 |         10
     1 |            2 |         11
     1 |            2 |         12
     1 |            2 |         13
     1 |            2 |         14
     1 |            3 |         15
     1 |            3 |         16
     1 |            3 |         17
     1 |             3 |         18
     1 |             3 |         19
     1 |             3 |         20
     1 |             3 |         21
     1 |             4 |         22
     1 |             4 |         23
     1 |             4 |         24
     1 |             4 |         25
     1 |             4 |         26
     1 |             4 |         27
     1 |             4 |         28
     1 |             5 |         29
     1 |             5 |         30
     2 |             5 |          1
     2 |             5 |          2
     2 |             5 |          3
     2 |             5 |          4
     2 |             5 |          5
     2 |             6 |          6
     2 |             6 |          7
     2 |             6 |          8
     2 |             6 |          9
     2 |             6 |         10
     2 |             6 |         11
     2 |             6 |         12
(42 rows)
```

- DENSE_RANK()

  Description: Generates consecutive sequence numbers for the values in each group. The same values have the same sequence number.

  Return type: bigint

  Example:

```
openGauss=# SELECT d_moy, d_fy_week_seq, dense_rank() OVER(PARTITION BY d_moy ORDER BY d_fy_week_seq) FROM tpcds.date_dim WHERE d_moy < 4 AND d_fy_week_seq < 7 ORDER BY 1,2;
 d_moy | d_fy_week_seq | dense_rank
-------+---------------+------------
     1 |             1 |          1
     1 |             1 |          1
     1 |             1 |          1
     1 |             1 |          1
     1 |             1 |          1
     1 |             1 |          1
     1 |             1 |          1
     1 |             2 |          2
     1 |             2 |          2
     1 |             2 |          2
     1 |             2 |          2
     1 |             2 |          2
     1 |             2 |          2
     1 |             2 |          2
     1 |             3 |          3
     1 |             3 |          3
     1 |             3 |          3
     1 |             3 |          3
     1 |             3 |          3
     1 |             3 |          3
     1 |             3 |          3
     1 |             4 |          4
     1 |             4 |          4
     1 |             4 |          4
     1 |             4 |          4
     1 |             4 |          4
     1 |             4 |          4
     1 |             4 |          4
     1 |             5 |          5
     1 |             5 |          5
     2 |             5 |          1
     2 |             5 |          1
     2 |             5 |          1
     2 |             5 |          1
     2 |             5 |          1
     2 |             6 |          2
     2 |             6 |          2
     2 |             6 |          2
     2 |             6 |          2
     2 |             6 |          2
     2 |             6 |          2
     2 |             6 |          2
(42 rows)
```

- PERCENT_RANK()

  Description: Generates corresponding sequence numbers for the values in each group. That is, the function calculates the value according to the formula Sequence number = (**rank** - 1)/(**total rows** - 1). **rank** is the corresponding sequence number generated based on the **RANK** function for the value and **totalrows** is the total number of elements in a group.

  Return type: double precision

  Example:

```
openGauss=# SELECT d_moy, d_fy_week_seq, percent_rank() OVER(PARTITION BY d_moy ORDER BY d_fy_week_seq) FROM tpcds.date_dim WHERE d_moy < 4 AND d_fy_week_seq < 7 ORDER BY 1,2;
 d_moy | d_fy_week_seq |   percent_rank
-------+---------------+------------------
     1 |             1 |                0
     1 |             1 |                0
     1 |             1 |                0
     1 |             1 |                0
     1 |             1 |                0
     1 |             1 |                0
     1 |             1 |                0
     1 |             2 | .241379310344828
     1 |             2 | .241379310344828
     1 |             2 | .241379310344828
     1 |             2 | .241379310344828
     1 |             2 | .241379310344828
     1 |             2 | .241379310344828
     1 |             2 | .241379310344828
     1 |             3 | .482758620689655
     1 |             3 | .482758620689655
     1 |             3 | .482758620689655
     1 |             3 | .482758620689655
     1 |             3 | .482758620689655
     1 |             3 | .482758620689655
     1 |             3 | .482758620689655
     1 |             4 | .724137931034483
     1 |             4 | .724137931034483
     1 |             4 | .724137931034483
     1 |             4 | .724137931034483
     1 |             4 | .724137931034483
     1 |             4 | .724137931034483
     1 |             4 | .724137931034483
     1 |             5 |  .96551724137931
     1 |             5 |  .96551724137931
     2 |             5 |                0
     2 |             5 |                0
     2 |             5 |                0
     2 |             5 |                0
     2 |             5 |                0
     2 |             6 | .454545454545455
     2 |             6 | .454545454545455
     2 |             6 | .454545454545455
     2 |             6 | .454545454545455
     2 |             6 | .454545454545455
     2 |             6 | .454545454545455
     2 |            6 | .454545454545455
(42 rows)
```

- CUME_DIST()

  Description: Generates accumulative distribution sequence numbers for the values in each group. That is, the function calculates the value according to the following formula: Sequence number = Number of rows preceding or peer with current row/Total rows.

  Return type: double precision

  Example:

```
openGauss=# SELECT d_moy, d_fy_week_seq, cume_dist() OVER(PARTITION BY d_moy ORDER BY
d_fy_week_seq) FROM tpcds.date_dim e_dim WHERE d_moy < 4 AND d_fy_week_seq < 7 ORDER BY
1,2;
 d_moy | d_fy_week_seq |    cume_dist
-------+---------------+------------------
     1 |             1 | .233333333333333
     1 |             1 | .233333333333333
     1 |             1 | .233333333333333
     1 |             1 | .233333333333333
     1 |             1 | .233333333333333
     1 |             1 | .233333333333333
     1 |             1 | .233333333333333
     1 |             2 | .466666666666667
     1 |             2 | .466666666666667
     1 |             2 | .466666666666667
     1 |             2 | .466666666666667
     1 |             2 | .466666666666667
     1 |             2 | .466666666666667
     1 |             2 | .466666666666667
     1 |             3 |               .7
     1 |             3 |               .7
     1 |             3 |               .7
     1 |             3 |               .7
     1 |             3 |               .7
     1 |             3 |               .7
     1 |             3 |               .7
     1 |             4 | .933333333333333
     1 |             4 | .933333333333333
     1 |             4 | .933333333333333
     1 |             4 | .933333333333333
     1 |             4 | .933333333333333
     1 |             4 | .933333333333333
     1 |             4 | .933333333333333
     1 |             5 |                1
     1 |             5 |                1
     2 |             5 | .416666666666667
     2 |             5 | .416666666666667
     2 |             5 | .416666666666667
     2 |             5 | .416666666666667
     2 |             5 | .416666666666667
     2 |             6 |                1
     2 |             6 |                1
     2 |             6 |                1
     2 |             6 |                1
     2 |             6 |                1
     2 |             6 |                1
     2 |             6 |                1
(42 rows)
```

- NTILE(num_buckets integer)

  Description: Equally allocates sequential data sets to the buckets whose quantity is specified by **num_buckets** according to **num_buckets integer** and allocates the bucket number to each row. Divide the partition as evenly as possible.

  Return type: integer

Example:

```
openGauss=# SELECT d_moy, d_fy_week_seq, ntile(3) OVER(PARTITION BY d_moy ORDER BY
d_fy_week_seq) FROM tpcds.date_dim WHERE d_moy < 4 AND d_fy_week_seq < 7 ORDER BY 1,2;
 d_moy | d_fy_week_seq | ntile
-------+---------------+-------
     1 |             1 |     1
     1 |             1 |     1
     1 |             1 |     1
     1 |             1 |     1
     1 |             1 |     1
     1 |             1 |     1
     1 |             1 |     1
     1 |             2 |     1
     1 |             2 |     1
     1 |             2 |     1
     1 |             2 |     2
     1 |             2 |     2
     1 |             2 |     2
     1 |             2 |     2
     1 |             3 |     2
     1 |             3 |     2
     1 |             3 |     2
     1 |             3 |     2
     1 |             3 |     2
     1 |             3 |     2
     1 |             3 |     3
     1 |             4 |     3
     1 |             4 |     3
     1 |             4 |     3
     1 |             4 |     3
     1 |             4 |     3
     1 |             4 |     3
     1 |             4 |     3
     1 |             5 |     3
     1 |             5 |     3
     2 |             5 |     1
     2 |             5 |     1
     2 |             5 |     1
     2 |             5 |     1
     2 |             5 |     2
     2 |             6 |     2
     2 |             6 |     2
     2 |             6 |     2
     2 |             6 |     3
     2 |             6 |     3
     2 |             6 |     3
     2 |             6 |     3
(42 rows)
```

- LAG(value any [, offset integer [, default any]])

  Description: Generates lag values for the corresponding values in each group. That is, the value of the row obtained by moving forward the row corresponding to the current value by **offset** (integer) is the sequence number. If the row does not exist after the moving, the result value is the default value. If omitted, **offset** defaults to **1** and **default** to **NULL**. The type of the **default** value must be the same as that of **value**.

  Return type: same as the parameter type

  Example:

```
openGauss=# SELECT d_moy, d_fy_week_seq, lag(d_moy,3,null) OVER(PARTITION BY d_moy ORDER
BY d_fy_week_seq) FROM tpcds.date_dim WHERE d_moy < 4 AND d_fy_week_seq < 7 ORDER BY 1,2;
 d_moy | d_fy_week_seq | lag
-------+---------------+-----
     1 |             1 |
     1 |             1 |
     1 |             1 |
     1 |             1 |    1
     1 |             1 |    1
     1 |             1 |    1
     1 |             1 |    1
     1 |             2 |    1
     1 |             2 |    1
     1 |             2 |    1
     1 |             2 |    1
     1 |             2 |    1
     1 |             2 |    1
     1 |             2 |    1
     1 |             3 |    1
     1 |             3 |    1
     1 |             3 |    1
     1 |             3 |    1
     1 |             3 |    1
     1 |             3 |    1
     1 |             3 |    1
     1 |             4 |    1
     1 |             4 |    1
     1 |             4 |    1
     1 |             4 |    1
     1 |             4 |    1
     1 |             4 |    1
     1 |             4 |    1
     1 |             5 |    1
     1 |             5 |    1
     2 |             5 |
     2 |             5 |
     2 |             5 |
     2 |             5 |    2
     2 |             5 |    2
     2 |             6 |    2
     2 |             6 |    2
     2 |             6 |    2
     2 |             6 |    2
     2 |             6 |    2
     2 |             6 |    2
     2 |             6 |    2
(42 rows)
```

- LEAD(value any [, offset integer [, default any]])

  Description: Generates leading values for the corresponding values in each group. That is, the value of the row obtained by moving backward the row corresponding to the current value by **offset** (integer) is the sequence number. If the row after the moving exceeds the total number of rows for the current group, the result value is the default value. If omitted, **offset** defaults to **1** and **default** to **NULL**. The type of the **default** value must be the same as that of the **value** value.

  Return type: same as the parameter type

  Example:

```
openGauss=# SELECT d_moy, d_fy_week_seq, lead(d_fy_week_seq,2) OVER(PARTITION BY d_moy
ORDER BY d_fy_week_seq) FROM  tpcds.date_dim WHERE d_moy < 4 AND d_fy_week_seq < 7 ORDER
BY 1,2;
d_moy | d_fy_week_seq | lead
-------+---------------+------
     1 |             1 |    1
     1 |             1 |    1
     1 |             1 |    1
     1 |             1 |    1
     1 |             1 |    1
     1 |             1 |    2
     1 |             1 |    2
     1 |             2 |    2
     1 |             2 |    2
     1 |             2 |    2
     1 |             2 |   2
     1 |             2 |   2
     1 |             2 |   3
     1 |             2 |   3
     1 |             3 |   3
     1 |             3 |   3
     1 |             3 |   3
     1 |             3 |   3
     1 |             3 |   3
     1 |             3 |   4
     1 |             3 |   4
     1 |             4 |   4
     1 |             4 |   4
     1 |             4 |   4
     1 |             4 |   4
     1 |             4 |   4
     1 |             4 |   5
     1 |             4 |   5
     1 |             5 |
     1 |             5 |
     2 |             5 |   5
     2 |             5 |   5
     2 |             5 |   5
     2 |             5 |   6
     2 |             5 |   6
     2 |             6 |   6
     2 |             6 |   6
     2 |             6 |   6
     2 |             6 |   6
     2 |             6 |   6
     2 |             6 |
     2 |             6 |
(42 rows)
```

- FIRST_VALUE(value any)

  Description: Returns the first value of each group.

  Return type: same as the parameter type

  Example:

```
openGauss=# SELECT d_moy, d_fy_week_seq, first_value(d_fy_week_seq) OVER(PARTITION BY d_moy
ORDER BY d_fy_week_seq) FROM tpcds.date_dim WHERE d_moy < 4 AND d_fy_week_seq < 7 ORDER
BY 1,2;
 d_moy | d_fy_week_seq | first_value
-------+---------------+-------------
     1 |             1 |           1
     1 |             1 |           1
     1 |             1 |           1
     1 |             1 |           1
     1 |             1 |           1
     1 |             1 |           1
     1 |             1 |           1
     1 |             2 |           1
     1 |             2 |           1
     1 |             2 |           1
     1 |             2 |           1
     1 |             2 |           1
     1 |             2 |           1
     1 |             2 |           1
     1 |             3 |           1
     1 |             3 |           1
     1 |             3 |           1
     1 |             3 |           1
     1 |             3 |           1
     1 |             3 |           1
     1 |             3 |           1
     1 |             4 |           1
     1 |             4 |           1
     1 |             4 |           1
     1 |             4 |           1
     1 |               4 |            1
     1 |               4 |            1
     1 |               4 |            1
     1 |               5 |            1
     1 |               5 |            1
     2 |               5 |            5
     2 |               5 |            5
     2 |               5 |            5
     2 |               5 |            5
     2 |               5 |            5
     2 |               6 |            5
     2 |               6 |            5
     2 |               6 |            5
     2 |               6 |            5
     2 |               6 |            5
     2 |               6 |            5
     2 |               6 |            5
(42 rows)
```

- LAST_VALUE(value any)

  Description: Returns the last value of each group.

  Return type: same as the parameter type

  Example:

```
openGauss=# SELECT d_moy, d_fy_week_seq, last_value(d_moy) OVER(PARTITION BY d_moy ORDER
BY d_fy_week_seq) FROM tpcds.date_dim WHERE d_moy < 4 AND d_fy_week_seq < 6 ORDER BY 1,2;
  d_moy | d_fy_week_seq | last_value
-------+---------------+------------
     1 |               1 |           1
     1 |               1 |           1
     1 |               1 |           1
     1 |               1 |           1
     1 |               1 |           1
     1 |               1 |           1
     1 |               1 |           1
     1 |               2 |           1
     1 |               2 |           1
     1 |               2 |           1
     1 |               2 |           1
     1 |               2 |           1
     1 |               2 |           1
     1 |               2 |           1
     1 |               3 |           1
     1 |               3 |           1
     1 |               3 |           1
     1 |               3 |           1
     1 |               3 |           1
     1 |               3 |           1
     1 |               3 |           1
     1 |               4 |           1
     1 |               4 |           1
     1 |               4 |           1
     1 |               4 |           1
     1 |               4 |           1
     1 |               4 |           1
     1 |               4 |           1
     1 |               5 |           1
     1 |               5 |           1
     2 |               5 |           2
     2 |               5 |           2
     2 |               5 |           2
     2 |               5 |           2
     2 |               5 |           2
(35 rows)
```

- DELTA

Description: Returns the difference between the current row and the previous row.

Parameter: numeric

Return type: numeric

- NTH_VALUE(value any, nth integer)

  Description: Returns the *n*th row for a group. If the row does not exist, **NULL** is returned by default.

  Return type: same as the parameter type

  Example:

```
openGauss=# SELECT d_moy, d_fy_week_seq, nth_value(d_fy_week_seq,6) OVER(PARTITION BY
d_moy ORDER BY d_fy_week_seq) FROM tpcds.date_dim WHERE d_moy < 4 AND d_fy_week_seq < 6
ORDER BY 1,2;
 d_moy | d_fy_week_seq | nth_value
-------+---------------+-----------
     1 |             1 |         1
     1 |             1 |         1
     1 |             1 |         1
     1 |             1 |         1
     1 |             1 |         1
     1 |             1 |         1
     1 |             1 |         1
     1 |             2 |         1
     1 |             2 |         1
     1 |             2 |         1
     1 |             2 |         1
     1 |             2 |         1
     1 |             2 |         1
     1 |             2 |         1
     1 |             3 |         1
     1 |             3 |         1
     1 |             3 |         1
     1 |             3 |         1
     1 |             3 |         1
     1 |             3 |         1
     1 |             3 |         1
     1 |             4 |         1
     1 |             4 |         1
     1 |             4 |         1
     1 |             4 |         1
     1 |             4 |         1
     1 |             4 |         1
     1 |             4 |         1
     1 |             5 |         1
     1 |             5 |         1
     2 |             5 |
     2 |             5 |
     2 |             5 |
     2 |             5 |
     2 |             5 |
(35 rows)
```

11.5.20 Security Functions

Security Functions

- gs_encrypt_aes128(encryptstr,keystr)

  Description: Encrypts **encryptstr** strings using **keystr** as the key and returns encrypted strings. The value of **keystr** ranges from 8 to 16 bytes and contains at least three types of the following characters: uppercase letters, lowercase letters, digits, and special characters.

Return type: text

Length of the return value: At least 92 bytes and no more than (4*[*Len*/3]+68) bytes, where *Len* indicates the length of the data before encryption (unit: byte).

Example:

```
openGauss=# SELECT gs_encrypt_aes128('MPPDB','Asdf1234');

                          gs_encrypt_aes128
-------------------------------------------------------------------------------------
gwditQLQG8NhFw4OuoKhhQJoXojhFlYkjeG0aYdSCtLCnIUgkNwvYI04KbuhmcGZp8jWizBdR1vU9CspjuzI0lbz12A=
(1 row)
```

NOTE

A decryption password is required during the execution of this function. For security purposes, the gsql tool does not record the SQL statements containing the function name in the execution history. That is, the execution history of this function cannot be found in gsql by paging up and down.

- gs_encrypt(encryptstr,keystr, encrypttype)

  Description: Encrypts **encryptstr** strings using **keystr** as the key and returns encrypted strings based on **encrypttype**. The value of **keystr** contains 8 to 16 bytes and at least three types of the following characters: uppercase letters, lowercase letters, digits, and special characters. The value of **encrypttype** can be **aes128** or **sm4**.

  Return type: text

  Example:

```
openGauss=#  SELECT gs_encrypt('MPPDB','Asdf1234','sm4');
        gs_encrypt
 ------------------------------
 ZBzOmaGA4Bb+coyucJ0B8AkIShqc
(1 row)
```

NOTE

A decryption password is required during the execution of this function. For security purposes, the gsql tool does not record the SQL statements containing the function name in the execution history. That is, the execution history of this function cannot be found in gsql by paging up and down.

- gs_decrypt_aes128(decryptstr,keystr)

  Description: Decrypts **decrypt** strings using **keystr** as the key and returns decrypted strings. The **keystr** used for decryption must be consistent with that used for encryption. **keystr** cannot be empty.

NOTE

This parameter needs to be used with the **gs_encrypt_aes128** encryption function.

Return type: text

Example:

```
openGauss=# SELECT
gs_decrypt_aes128('gwditQLQG8NhFw4OuoKhhQJoXojhFlYkjeG0aYdSCtLCnIUgkNwvYI04KbuhmcGZp8jWizBdR1vU9CspjuzI0lbz12A=','1234');
 gs_decrypt_aes128
-------------------
 MPPDB
(1 row)
```

NOTE

A decryption password is required during the execution of this function. For security purposes, the gsql tool does not record the SQL statements containing the function name in the execution history. That is, the execution history of this function cannot be found in gsql by paging up and down.

- gs_decrypt(decryptstr, keystr, decrypttype)

  Description: Decrypts **decrypt** strings using **keystr** as the key and returns decrypted strings based on **decrypttype**. The **decrypttype** and **keystr** used for decryption must be consistent with those used for encryption. The value of **keystr** cannot be empty. The value of **decrypttype** can be **aes128** or **sm4**.

  This function needs to be used with the **gs_encrypt** encryption function.

  Return type: text

  Example:

```
openGauss=# select gs_decrypt('ZBzOmaGA4Bb+coyucJ0B8AkIShqc','Asdf1234','sm4');
 gs_decrypt
------------
 MPPDB
(1 row)
```

  NOTE

  A decryption password is required during the execution of this function. For security purposes, the gsql tool does not record the SQL statements containing the function name in the execution history. That is, the execution history of this function cannot be found in gsql by paging up and down.

- gs_password_deadline

  Description: Indicates the number of remaining days before the password of the current user expires.

  Return type: interval

  Example:

```
openGauss=# SELECT gs_password_deadline();
  gs_password_deadline
-------------------------
 83 days 17:44:32.196094
(1 row)
```

- gs_password_notifytime()

  Description: Specifies the number of days prior to password expiration that a user will receive a reminder.

  Return type: int32

- login_audit_messages(BOOLEAN)

  Description: Queries login information about a login user.

  Return type: tuple

  Example:

  - Check the date, time, and IP address of the last successful login.

```
openGauss=> select * from login_audit_messages(true);
 username | database |       logintime        |    mytype     | result | client_conninfo
----------+----------+------------------------+---------------+--------+-----------------
 omm      | openGauss | 2020-06-29 21:56:40+08 | login_success | ok     | gsql@[local]
(1 row)
```

  - Check the number, date, and time of failed attempts since the previous successful login.

```
openGauss=>  select * from login_audit_messages(false);
 username | database |       logintime        |    mytype    | result |  client_conninfo
----------+----------+------------------------+--------------+--------+-------------------
 omm      | openGauss | 2020-06-29 21:57:55+08 | login_failed | failed | [unknown]@[local]
 omm      | openGauss | 2020-06-29 21:57:53+08 | login_failed | failed | [unknown]@[local]
(2 rows)
```

- login_audit_messages_pid

  Description: Queries login information about a login user. Different from **login_audit_messages**, this function queries login information based on **backendid**. Information about subsequent logins of the same user does not alter the query result of previous logins and cannot be found using this function.

  Return type: tuple

  NOTE

  When the thread pool is enabled, **backendid** obtained in the same session may change due to thread switchover. As a result, the return values are different when the function is called for multiple times. You are not advised to call this function when the thread pool is enabled.

  Example:

  – Check the date, time, and IP address of the last successful login.

```
openGauss=> SELECT * FROM login_audit_messages_pid(true);
 username | database |       logintime        |    mytype     | result | client_conninfo |    backendid
----------+----------+------------------------+---------------+--------+-----------------+-----------------
 omm      | openGauss | 2020-06-29 21:56:40+08 | login_success | ok     | gsql@[local]    |
139823109633792
(1 row)
```

  – Check the number, date, and time of failed attempts since the previous successful login.

```
openGauss=> SELECT * FROM login_audit_messages_pid(false);
 username | database |       logintime        |    mytype    | result |  client_conninfo  |    backendid
----------+----------+------------------------+--------------+--------+-------------------+-----------------
 omm      | openGauss | 2020-06-29 21:57:55+08 | login_failed | failed | [unknown]@[local] |
139823109633792
 omm      | openGauss | 2020-06-29 21:57:53+08 | login_failed | failed | [unknown]@[local] |
139823109633792
(2 rows)
```

- inet_server_addr

  Description: Displays the server IP address.

  Return type: inet

  Example:

```
openGauss=# SELECT inet_server_addr();
 inet_server_addr
------------------
 10.10.0.13
(1 row)
```

  NOTE

  - The client IP address 10.10.0.50 and server IP address 10.10.0.13 are used as an example.
  - If the database is connected to the local PC, the value is empty.

- inet_client_addr

  Description: Displays the client IP address.

  Return type: inet

  Example:

```
openGauss=# SELECT inet_client_addr();
 inet_client_addr
------------------
 10.10.0.50
(1 row)
```

NOTE

- The client IP address 10.10.0.50 and server IP address 10.10.0.13 are used as an example.
- If the database is connected to the local PC, the value is empty.

- pg_query_audit

  Description: Views audit logs of the primary database node.

  Return type: record

  The following table describes return fields.

Name	Type	Description
time	timestamp with time zone	Operation time
type	text	Operation
result	text	Operation result
userid	oid	User ID
username	text	Name of the user who performs the operation
database	text	Database name
client_conninfo	text	Client connection information
object_name	text	Object name
detail_info	text	Operation details
node_name	text	Node name
thread_id	text	Thread ID
local_port	text	Local port
remote_port	text	Remote port

  For details about how to use the function and details about function examples, see **Querying Audit Results**.

- pg_delete_auditDescription: Deletes audit logs in a specified period. Return type: void For details about how to use the function and details about function examples, see Maintaining Audit Logs.

- alldigitsmasking

Description: Specifies the internal function of the masking policy, which is used to anonymize all characters.

Parameter: col text, letter character default '0'

Return type: text

- creditcardmasking

  Description: Specifies the internal function of the masking policy, which is used to anonymize all credit card information.

  Parameter: col text, letter character default 'x'

  Return type: text

- randommasking

  Description: Specifies the internal function of the masking policy. The random policy is used.

  Parameter: col text

  Return type: text

- fullemailmasking

  Description: Specifies the internal function of the masking policy, which is used to anonymize the text (except @) before the last period (.).

  Parameter: col text, letter character default 'x'

  Return type: text

- basicemailmasking

  Description: Specifies the internal function of the masking policy, which is used to anonymize the text before the first at sign (@).

  Parameter: col text, letter character default 'x'

  Return type: text

- shufflemasking

  Description: Specifies the internal function of the masking policy, which is used to sort characters out of order.

  Parameter: col text

  Return type: text

11.5.21 Ledger Database Functions

The current feature is a lab feature. Contact Huawei technical support before using it.

- get_dn_hist_relhash(text, text)

  Description: Returns the hash value of table-level data in a specified tamper-proof user table. This function is used only in distributed mode.

  Parameter type: text

  Return type: hash16

- ledger_hist_check(text, text)

  Description: Verifies the consistency between the hash value of table-level data in a specified tamper-proof user table and that in the corresponding history table.

  Parameter type: text

Return type: Boolean

- ledger_hist_repair(text, text)

  Description: Restores the hash value of the history table corresponding to the specified tamper-proof user table to be the same as that of the user table, and returns the hash difference.

  Parameter type: text

  Return type: hash16

- ledger_hist_archive(text, text)

  Description: Archives the history table corresponding to a specified tamper-proof user table to the **hist_back** folder in the audit log directory.

  Parameter type: text

  Return type: Boolean

- ledger_gchain_check(text, text)

  Description: Verifies the consistency between the history table hash corresponding to the specified tamper-proof user table and the **relhash** corresponding to the global history table.

  Parameter type: text

  Return type: Boolean

- ledger_gchain_repair(text, text)

  Description: Restores **relhash** of a specified tamper-proof user table in the global history table so that the hash is the same as that in the history table, and returns the hash difference.

  Parameter type: text

  Return type: hash16

- ledger_gchain_archive(void)

  Description: Archives global history tables to the **hist_back** folder in the audit log directory.

  Parameter type: void

  Return type: Boolean

- hash16in(cstring)

  Description: Converts the input hexadecimal string into the internal hash16 format.

  Parameter type: cstring

  Return type: hash16

- hash16out(hash16)

  Description: Converts internal hash16 data to hexadecimal cstring data.

  Parameter type: hash16

  Return type: cstring

- hash32in(cstring)

  Description: Converts the input hexadecimal string (32 characters) into the internal type hash32.

  Parameter type: cstring

  Return type: hash32

- hash32out(hash32)

  Description: Converts internal hash32 data to hexadecimal cstring data.

  Parameter type: cstring

  Return type: hash32

11.5.22 Encrypted Equality Functions

- byteawithoutorderwithequalcolin(cstring)

  Description: Converts input data to the internal byteawithoutorderwithequalcol format.

  Parameter type: cstring

  Return type: byteawithoutorderwithequalcol
- byteawithoutorderwithequalcolout(byteawithoutorderwithequalcol)

  Description: Converts internal data of the byteawithoutorderwithequalcol type to data of the cstring type.

  Parameter type: byteawithoutorderwithequalcol

  Return type: cstring
- byteawithoutorderwithequalcolsend(byteawithoutorderwithequalcol)

  Description: Converts data of the byteawithoutorderwithequalcol type to data of the bytea type.

  Parameter type: byteawithoutorderwithequalcol

  Return type: bytea
- byteawithoutorderwithequalcolrecv(internal)

  Description: Converts data of the byteawithoutorderwithequalcol type to data of the byteawithoutorderwithequalcol type.

  Parameter type: internal

  Return type: byteawithoutorderwithequalcol
- byteawithoutorderwithequalcoltypmodin(_cstring)

  Description: Converts data of the byteawithoutorderwithequalcol type to data of the byteawithoutorderwithequalcol type.

  Parameter type: _cstring

  Return type: int4
- byteawithoutorderwithequalcoltypmodout(int4)

  Description: Converts data of the int4 type into data of the cstring type.

  Parameter type: int4

  Return type: cstring
- byteawithoutordercolin(cstring)

  Description: Converts input data to the internal byteawithoutordercolin format.

  Parameter type: cstring

  Return type: byteawithoutordercol
- byteawithoutordercolout(byteawithoutordercol)

  Description: Converts internal data of the byteawithoutordercol type to data of the cstring type.

Parameter type: byteawithoutordercol

Return type: cstring

- byteawithoutordercolsend(byteawithoutordercol)

  Description: Converts data of the byteawithoutordercol type to data of the bytea type.

  Parameter type: byteawithoutordercol

  Return type: bytea

- byteawithoutordercolrecv(internal)

  Description: Converts data of the byteawithoutordercol type to data of the byteawithoutordercol type.

  Parameter type: internal

  Return type: byteawithoutordercol

- byteawithoutorderwithequalcolcmp(byteawithoutorderwithequalcol, byteawithoutorderwithequalcol)

  Description: Compares two byteawithoutorderwithequalcol data sizes. If the first data size is smaller than the second one, **–1** is returned. If the first data size is equal to the second one, **0** is returned. If the first data size is larger than the second one, **1** is returned.

  Parameter type: byteawithoutorderwithequalcol, byteawithoutorderwithequalcol

  Return type: int4

- byteawithoutorderwithequalcolcmpbytear(byteawithoutorderwithequalcol, bytea)

  Description: Compares the byteawithoutorderwithequalcol and bytea data sizes. If the first data size is smaller than the second one, **–1** is returned. If the first data size is equal to the second one, **0** is returned. If the first data size is larger than the second one, **1** is returned.

  Parameter type: byteawithoutorderwithequalcol, bytea

  Return type: int4

- byteawithoutorderwithequalcolcmpbyteal(bytea, byteawithoutorderwithequalcol)

  Description: Compares the bytea and byteawithoutorderwithequalcol data sizes. If the first data size is smaller than the second one, **–1** is returned. If the first data size is equal to the second one, **0** is returned. If the first data size is larger than the second one, **1** is returned.

  Parameter type: byteawithoutorderwithequalcol, bytea

  Return type: int4

- byteawithoutorderwithequalcoleq(byteawithoutorderwithequalcol, byteawithoutorderwithequalcol)

  Description: Compares two byteawithoutorderwithequalcol data records. If they are the same, **true** is returned. Otherwise, **false** is returned.

  Parameter type: byteawithoutorderwithequalcol, bytea

  Return type: Boolean

- byteawithoutorderwithequalcoleqbyteal(bytea, byteawithoutorderwithequalcol)

Description: Compares the bytea and byteawithoutorderwithequalcol data records. If they are the same, **true** is returned. Otherwise, **false** is returned.

Parameter type: bytea, byteawithoutorderwithequalcol

Return type: Boolean

- byteawithoutorderwithequalcoleqbytear(byteawithoutorderwithequalcol, bytea)

  Description: Compares the byteawithoutorderwithequalcol and bytea data records. If they are the same, **true** is returned. Otherwise, **false** is returned.

  Parameter type: byteawithoutorderwithequalcol, bytea

  Return type: Boolean

- byteawithoutorderwithequalcolne(byteawithoutorderwithequalcol, byteawithoutorderwithequalcol)

  Description: Compares two byteawithoutorderwithequalcol data records. If they are different, **true** is returned. Otherwise, **false** is returned.

  Parameter type: byteawithoutorderwithequalcol, byteawithoutorderwithequalcol

  Return type: Boolean

- byteawithoutorderwithequalcolnebyteal(bytea, byteawithoutorderwithequalcol)

  Description: Compares the bytea and byteawithoutorderwithequalcol data records. If they are the same, **true** is returned. Otherwise, **false** is returned.

  Parameter type: bytea, byteawithoutorderwithequalcol

  Return type: Boolean

- byteawithoutorderwithequalcolnebytear(byteawithoutorderwithequalcol, bytea)

  Description: Compares the byteawithoutorderwithequalcol and bytea data records. If they are the same, **true** is returned. Otherwise, **false** is returned.

  Parameter type: byteawithoutorderwithequalcol, bytea

  Return type: Boolean

- hll_hash_byteawithoutorderwithequalcol(byteawithoutorderwithequalcol)

  Description: Returns the hll hash value of byteawithoutorderwithequalcol.

  Parameter type: byteawithoutorderwithequalcol

  Return type: hll_hashval

Example

Functions such as byteawithoutorderwithequalcolin and byteawithoutorderwithequalcolout are read/write format conversion functions such as in, out, send, and recv specified by the data type byteawithoutorderwithequalcol in the database kernel. For details, see the byteain and byteaout functions of the bytea type. However, the local CEK must be verified, and the function can be successfully executed only when the encrypted column contains a CEK OID that can be found on the local host.

```
-- In this example, there is an encrypted table int_type, and int_col2 is the encrypted column.

-- Use a non-encrypted client to connect to the database and query the ciphertext of the encrypted column.
```

```
openGauss=# select int_col2 from int_type;
                                                          int_col2
------------------------------------------------------------------------------------------------------------------------
------------------------

\x01c35301bf421c8edf38c34704bcc82838742917778ccb402a1b7452ad4a6ac7371acc0ac33100000035fe3424
919854c86194f1aa5bb4e1ca656e8fc6d05324a1419b69f488bdc3c6
(1 row)

-- The ciphertext of the encrypted column is used as the input parameter of
byteawithoutorderwithequalcolin. The format is converted from cstring to byteawithoutorderwithequalcol.
openGauss=# select
byteawithoutorderwithequalcolin('\x01c35301bf421c8edf38c34704bcc82838742917778ccb402a1b7452ad4a
6ac7371acc0ac33100000035fe3424919854c86194f1aa5bb4e1ca656e8fc6d05324a1419b69f488bdc3c6');
                                                 byteawithoutorderwithequalcolin
------------------------------------------------------------------------------------------------------------------------
------------------------

\x01c35301bf421c8edf38c34704bcc82838742917778ccb402a1b7452ad4a6ac7371acc0ac33100000035fe3424
919854c86194f1aa5bb4e1ca656e8fc6d05324a1419b69f488bdc3c6
(1 row)
```

Implementations of functions such as byteawithoutorderwithequalcolin search for CEK and determine whether it is a normal encrypted data type.

If the format of the data entered by the user is not the encrypted data format and the corresponding CEK cannot be found on the local host, an error is returned.

```
openGauss=# SELECT * FROM
byteawithoutorderwithequalcolsend('\x907219912381298461289346129'::byteawithoutorderwithequalcol);
ERROR:  cek with OID 596711794 not found
LINE 1: SELECT * FROM byteawithoutorderwithequalcolsend('\x907219912...
                                                 ^
openGauss=# SELECT * FROM
byteawithoutordercolout('\x90721901999999999999912381298461289346129');
ERROR:  cek with OID 2566986098 not found
LINE 1: SELECT * FROM byteawithoutordercolout('\x9072190199999999999...

SELECT * FROM
byteawithoutorderwithequalcolrecv('\x90721901999999999999912381298461289346129'::byteawithoutorde
rwithequalcol);
ERROR:  cek with OID 2566986098 not found
                                      ^
openGauss=# SELECT * FROM
byteawithoutorderwithequalcolsend('\x90721901999999999999912381298461289346129'::byteawithoutorde
rwithequalcol);
ERROR:  cek with OID 2566986098 not found
LINE 1: SELECT * FROM byteawithoutorderwithequalcolsend('\x907219019...
                                                 ^
```

11.5.23 Set Returning Functions

Series Generating Functions

- generate_series(start, stop)

  Description: Generates a series of values, from **start** to **stop** with a step size of one.

  Parameter type: int, bigint, numeric

  Return type: setof int, setof bigint, setof numeric (same as the parameter type)

- generate_series(start, stop, step)

  Description: Generates a series of values, from **start** to **stop** with a step size of **step**.

Parameter type: int, bigint, numeric

Return type: setof int, setof bigint, setof numeric (same as the parameter type)

- generate_series(start, stop, step interval)

  Description: Generates a series of values, from **start** to **stop** with a step size of **step**.

  Parameter type: timestamp or timestamp with time zone

  Return type: setof timestamp or setof timestamp with time zone (same as parameter type)

When **step** is positive, zero rows are returned if **start** is greater than **stop**. Conversely, when **step** is negative, zero rows are returned if **start** is less than **stop**. Zero rows are also returned for **NULL** inputs. It is an error for **step** to be zero.

Example:

```
openGauss=# SELECT * FROM generate_series(2,4);
 generate_series
-----------------
               2
               3
               4
(3 rows)

openGauss=# SELECT * FROM generate_series(5,1,-2);
 generate_series
-----------------
               5
               3
               1
(3 rows)

openGauss=# SELECT * FROM generate_series(4,3);
 generate_series
-----------------
(0 rows)

-- This example applies to the date-plus-integer operator.
openGauss=# SELECT current_date + s.a AS dates FROM generate_series(0,14,7) AS s(a);
   dates
------------
 2017-06-02
 2017-06-09
 2017-06-16
(3 rows)

openGauss=# SELECT * FROM generate_series('2008-03-01 00:00'::timestamp, '2008-03-04 12:00', '10
hours');
   generate_series
---------------------
 2008-03-01 00:00:00
 2008-03-01 10:00:00
 2008-03-01 20:00:00
 2008-03-02 06:00:00
 2008-03-02 16:00:00
 2008-03-03 02:00:00
 2008-03-03 12:00:00
 2008-03-03 22:00:00
 2008-03-04 08:00:00
(9 rows)
```

Subscript Generating Functions

- generate_subscripts(array anyarray, dim int)

  Description: Generates a series comprising the given array's subscripts.

  Return type: setof int

- generate_subscripts(array anyarray, dim int, reverse boolean)

  Description: Generates a series comprising the given array's subscripts. When **reverse** is true, the series is returned in reverse order.

  Return type: setof int

generate_subscripts is a function that generates the set of valid subscripts for the specified dimension of the given array. Zero rows are returned for arrays that do not have the requested dimension, or for NULL arrays (but valid subscripts are returned for NULL array elements). Example:

```
-- Basic usage
openGauss=# SELECT generate_subscripts('{NULL,1,NULL,2}'::int[], 1) AS s;
 s
---
 1
 2
 3
 4
(4 rows)
-- Unnest a 2D array:
openGauss=# CREATE OR REPLACE FUNCTION unnest2(anyarray)
RETURNS SETOF anyelement AS $$
SELECT $1[i][j]
   FROM generate_subscripts($1,1) g1(i),
        generate_subscripts($1,2) g2(j);
$$ LANGUAGE sql IMMUTABLE;

openGauss=# SELECT * FROM unnest2(ARRAY[[1,2],[3,4]]);
 unnest2
---------
       1
       2
       3
       4
(4 rows)

-- Delete the function.
openGauss=# DROP FUNCTION unnest2;
```

11.5.24 Conditional Expression Functions

Conditional Expression Functions

- coalesce(expr1, expr2, ..., exprn)

  Description:

  Returns the first of its parameters that are not null.

  COALESCE(expr1, expr2) is equivalent to **CASE WHEN expr1 IS NOT NULL THEN expr1 ELSE expr2 END**.

  Example:

  ```
  openGauss=# SELECT coalesce(NULL,'hello');
   coalesce
  ----------
   hello
  (1 row)
  ```

Note:

- If all the expressions are equivalent to NULL in the expression list, this function returns **NULL**.
- This value is replaced by the default value when data is displayed.
- Like a **CASE** expression, **COALESCE** only evaluates the parameters that are needed to determine the result. That is, parameters to the right of the first not-**NULL** parameter are not evaluated.

- decode(base_expr, compare1, value1, Compare2,value2, ... default)

  Description: Compares **base_expr** with each **compare(n)** and **returns value(n)** if they are matched. If **base_expr** does not match each **compare(n)**, the default value is returned.

  Example:

```
openGauss=# SELECT decode('A','A',1,'B',2,0);
 case
------
 1
(1 row)
```

- nullif(expr1, expr2)

  Description: Returns **NULL** only when **expr1** is equal to **expr2**. Otherwise, **expr1** is returned.

  nullif(expr1, expr2) is equivalent to **CASE WHEN expr1 = expr2 THEN NULL ELSE expr1 END**.

  Example:

```
openGauss=# SELECT nullif('hello','world');
 nullif
--------
 hello
(1 row)
```

  Note:

  Assume the two parameter data types are different:

  - If implicit conversion exists between the two data types, implicitly convert the parameter of lower priority to this data type using the data type of higher priority. If the conversion succeeds, computation is performed. Otherwise, an error is returned. Example:

```
openGauss=# SELECT nullif('1234'::VARCHAR,123::INT4);
 nullif
--------
   1234
(1 row)
openGauss=# SELECT nullif('1234'::VARCHAR,'2012-12-24'::DATE);
ERROR:  invalid input syntax for type timestamp: "1234"
```

  - If implicit conversion is not applied between two data types, an error is returned. Example:

```
openGauss=# SELECT nullif(TRUE::BOOLEAN,'2012-12-24'::DATE);
ERROR:  operator does not exist: boolean = timestamp without time zone
LINE 1: SELECT nullif(TRUE::BOOLEAN,'2012-12-24'::DATE) FROM sys_dummy;
^
HINT:  No operator matches the given name and argument type(s). You might need to add
explicit type casts.
```

- nvl(expr1 , expr2)

  Description:

  - If **expr1** is **NULL**, **expr2** is returned.

– If **expr1** is not **NULL**, **expr1** is returned.

Example:

```
openGauss=# SELECT nvl('hello','world');
  nvl
-------
 hello
(1 row)
```

Note: Parameters **expr1** and **expr2** can be of any data type. If **expr1** and **expr2** are of different data types, NVL checks whether **expr2** can be implicitly converted to **expr1**. If it can, the data type of **expr1** is returned. Otherwise, an error is returned.

- greatest(expr1 [, ...])

  Description: Selects the largest value from a list of any number of expressions.

  Return type:

  Example:

```
openGauss=# SELECT greatest(1*2,2-3,4-1);
 greatest
----------
        3
(1 row)
openGauss=# SELECT greatest('HARRY', 'HARRIOT', 'HAROLD');
 greatest
----------
 HARRY
(1 row)
```

- least(expr1 [, ...])

  Description: Selects the smallest value from a list of any number of expressions.

  Example:

```
openGauss=# SELECT least(1*2,2-3,4-1);
 least
-------
    -1
(1 row)
openGauss=# SELECT least('HARRY','HARRIOT','HAROLD');
 least
--------
 HAROLD
(1 row)
```

- EMPTY_BLOB()

  Description: Initiates a BLOB variable in an **INSERT** or an **UPDATE** statement to a **NULL** value.

  Return type: BLOB

  Example:

```
-- Create a table.
openGauss=# CREATE TABLE blob_tb(b blob,id int);
-- Insert data.
openGauss=# INSERT INTO blob_tb VALUES (empty_blob(),1);
--Delete the table.
openGauss=# DROP TABLE blob_tb;
```

  Note: The length is 0 obtained using **DBE_LOB.GET_LENGTH**.

11.5.25 System Information Functions

Session Information Functions

- current_catalog

  Description: Name of the current database (called "catalog" in the SQL standard)

  Return type: name

  Example:

```
openGauss=# SELECT current_catalog;
 current_database
------------------
 openGauss
(1 row)
```

- current_database()

  Description: Name of the current database

  Return type: name

  Example:

```
openGauss=# SELECT current_database();
 current_database
------------------
 openGauss
(1 row)
```

- current_query()

  Description: Text of the currently executing query, as committed by the client (might contain more than one statement)

  Return type: text

  Example:

```
openGauss=# SELECT current_query();
      current_query
-------------------------
 SELECT current_query();
(1 row)
```

- current_schema[()]

  Description: Name of current schema

  Return type: name

  Example:

```
openGauss=# SELECT current_schema();
 current_schema
----------------
 public
(1 row)
```

  Remarks: **current_schema** returns the first valid schema name in the search path. (If the search path is empty or contains no valid schema name, **NULL** is returned.) This is the schema that will be used for any tables or other named objects that are created without specifying a target schema.

- current_schemas(Boolean)

  Description: Names of schemas in search path

  Return type: name[]

  Example:

```
openGauss=# SELECT current_schemas(true);
   current_schemas
---------------------
 {pg_catalog,public}
(1 row)
```

Note:

current_schemas(Boolean) returns an array of the names of all schemas presently in the search path. The Boolean option determines whether implicitly included system schemas such as **pg_catalog** are included in the returned search path.

NOTE

The search path can be altered at run time by running the following command:

```
SET search_path TO schema [, schema, ...]
```

- current_user

  Description: User name of current execution context

  Return type: name

  Example:

```
openGauss=# SELECT current_user;
 current_user
--------------
 omm
(1 row)
```

  Note: **current_user** is the user identifier that is applicable for permission checking. Normally it is equal to the session user, but it can be changed with **SET ROLE**. It also changes during the execution of functions with the attribute **SECURITY DEFINER**.

- definer_current_user

  Description: User name of current execution context

  Return type: name

  Example:

```
openGauss=# SELECT definer_current_user();
 definer_current_user
----------------------
 omm
(1 row)
```

- pg_current_sessionid()

  Description: Session ID of the current execution context

  Return type: text

  Example:

```
openGauss=# SELECT pg_current_sessionid();
    pg_current_sessionid
----------------------------
 1579228402.140190434944768
(1 row)
```

  Note: **pg_current_sessionid()** is used to obtain the session ID in the current execution context. The structure of the value is *Timestamp*. *Session ID*. When **enable_thread_pool** is set to **off**, the actual session ID is the thread ID.

- pg_current_sessid

  Description: Session ID of the current execution context

  Return type: text

Example:

```
openGauss=# select pg_current_sessid();
pg_current_sessid
-------------------
140308875015936
(1 row)
```

Note: In thread pool mode, the session ID of the current session is obtained. In non-thread pool mode, the background thread ID of the current session is obtained.

- pg_current_userid

  Description: Current user ID.

  Return type: text

```
openGauss=# SELECT pg_current_userid();
pg_current_userid
-------------------
10
(1 row)
```

- working_version_num()

  Description: Returns a version number regarding system compatibility.

  Return type: int

  Example:

```
openGauss=# SELECT working_version_num();
 working_version_num
---------------------
               92231
(1 row)
```

- tablespace_oid_name()

  Description: Queries the tablespace name based on the tablespace OID.

  Return type: text

  Example:

```
openGauss=# select tablespace_oid_name(1663);
 tablespace_oid_name
---------------------
 pg_default
(1 row)
```

- inet_client_addr()

  Description: Remote connection address. **inet_client_addr** returns the IP address of the current client.

  NOTE

  It is available only in remote connection mode.

  Return type: inet

  Example:

```
openGauss=# SELECT inet_client_addr();
 inet_client_addr
------------------
 10.10.0.50
(1 row)
```

- inet_client_port()

  Description: Remote connection port. **inet_client_port** returns the port number of the current client.

NOTE

It is available only in remote connection mode.

Return type: int

Example:

```
openGauss=# SELECT inet_client_port();
 inet_client_port
------------------
            33143
(1 row)
```

- inet_server_addr()

  Description: Local connection address. **inet_server_addr** returns the IP address on which the server accepted the current connection.

  NOTE

  It is available only in remote connection mode.

  Return type: inet

  Example:

```
openGauss=# SELECT inet_server_addr();
 inet_server_addr
------------------
 10.10.0.13
(1 row)
```

- inet_server_port()

  Description: Local connection port. **inet_server_port** returns the port number. All these functions return NULL if the current connection is via a Unix-domain socket.

  NOTE

  It is available only in remote connection mode.

  Return type: int

  Example:

```
openGauss=# SELECT inet_server_port();
 inet_server_port
------------------
             8000
(1 row)
```

- pg_backend_pid()

  Description: Process ID of the server process attached to the current session

  Return type: int

  Example:

```
openGauss=# SELECT pg_backend_pid();
 pg_backend_pid
-----------------
 140229352617744
(1 row)
```

- pg_conf_load_time()

  Description: Configures load time. **pg_conf_load_time** returns the timestamp with time zone when the server configuration files were last loaded.

  Return type: timestamp with time zone

  Example:

```
openGauss=# SELECT pg_conf_load_time();
      pg_conf_load_time
------------------------------
 2017-09-01 16:05:23.89868+08
(1 row)
```

- pg_my_temp_schema()

  Description: OID of the temporary schema of a session. The value is **0** if the OID does not exist.

  Return type: oid

  Example:

```
openGauss=# SELECT pg_my_temp_schema();
 pg_my_temp_schema
-------------------
                 0
(1 row)
```

  Note: **pg_my_temp_schema** returns the OID of the current session's temporary schema, or zero if it has none (because it has not created any temporary tables). **pg_is_other_temp_schema** returns true if the given OID is the OID of another session's temporary schema.

- pg_is_other_temp_schema(oid)

  Description: Specifies whether the schema is the temporary schema of another session.

  Return type: Boolean

  Example:

```
openGauss=# SELECT pg_is_other_temp_schema(25356);
 pg_is_other_temp_schema
-------------------------
 f
(1 row)
```

- pg_listening_channels()

  Description: Channel names that the session is currently listening on

  Return type: SETOF text

  Example:

```
openGauss=# SELECT pg_listening_channels();
 pg_listening_channels
-----------------------
(0 rows)
```

  Note: **pg_listening_channels** returns a set of names of channels that the current session is listening to.

- pg_postmaster_start_time()

  Description: Server start time **pg_postmaster_start_time** returns the **timestamp with time zone** when the server started.

  Return type: timestamp with time zone

  Example:

```
openGauss=# SELECT pg_postmaster_start_time();
   pg_postmaster_start_time
------------------------------
 2017-08-30 16:02:54.99854+08
(1 row)
```

- pg_get_ruledef(rule_oid)

  Description: Obtains the **CREATE RULE** command for a rule.

Return type: text

Example:

```
openGauss=# select * from pg_get_ruledef(24828);
                             pg_get_ruledef
-------------------------------------------------------------------
 CREATE RULE t1_ins AS ON INSERT TO t1 DO INSTEAD INSERT INTO t2 (id) VALUES (new.id);
(1 row)
```

- sessionid2pid()

  Description: Obtains PID information from a session ID (for example, the **sessid** column in **gs_session_stat**).

  Return type: int8

  Example:

```
openGauss=# select sessionid2pid(sessid::cstring) from gs_session_stat limit 2;
  sessionid2pid
-----------------
 139973107902208
 139973107902208
(2 rows)
```

- session_context('namespace' , 'parameter')

  Description: Obtains and returns the parameter values of a specified namespace.

  Return type: VARCHAR

  Example:

```
openGauss=# SELECT session_context('USERENV', 'CURRENT_SCHEMA');
session_context
------------------
  public
(1 row)
```

  Note: Currently, the **current_user**, **current_schema**, **client_info**, **ip_address**, **sessionid**, and **sid** parameters are supported.

- pg_trigger_depth()

  Description: Current nesting level of triggers

  Return type: int

  Example:

```
openGauss=# SELECT pg_trigger_depth();
 pg_trigger_depth
------------------
                0
(1 row)
```

- session_user

  Description: Session user name

  Return type: name

  Example:

```
openGauss=# SELECT session_user;
 session_user
--------------
 omm
(1 row)
```

  Note: **session_user** is usually the user who initiated the current database connection, but administrators can change this setting with **SET SESSION AUTHORIZATION**.

- user

  Description: Equivalent to **current_user**.

  Return type: name

  Example:

```
openGauss=# SELECT user;
 current_user
--------------
 omm
(1 row)
```

- getpgusername()

  Description: Obtains the database username.

  Return type: name

  Example:

```
openGauss=# select getpgusername();
 getpgusername
---------------
 GaussDB_userna
(1 row)
```

- getdatabaseencoding()

  Description: Obtains the database encoding mode.

  Return type: name

  Example:

```
openGauss=# select getdatabaseencoding();
 getdatabaseencoding
---------------------
 SQL_ASCII
(1 row)
```

- version()

  Description: Version information. **version** returns a string describing a server's version.

  Return type: text

  Example:

```
openGauss=# select version();
version
--------------------------------------------------------------------------------------------------------------------------
-------------------------------------
(GaussDB Kernel VxxxRxxxCxx build fab4f5ea) compiled at 2021-10-24 11:58:22 commit 3086 last mr
6592 release
(1 row)
```

- opengauss_version()

  Description: openGauss version information

  Return type: text

  Example:

```
openGauss=# select opengauss_version();
 opengauss_version
-------------------
 2.0.0
(1 row)
```

- gs_deployment()

  Description: Information about the deployment mode of the current system

  Return type: text

Example:

```
openGauss=# select gs_deployment();
   gs_deployment
---------------------
 BusinessCentralized
(1 row)
```

- get_hostname()

  Description: Returns the host name of the current node.

  Return type: text

  Example:

```
openGauss=# SELECT get_hostname();
 get_hostname
--------------
 linux-user
(1 row)
```

- get_nodename()

  Description: Returns the name of the current node.

  Return type: text

  Example:

```
openGauss=# SELECT get_nodename();
 get_nodename
--------------
 datanode1
(1 row)
```

- get_schema_oid(cstring)

  Description: Returns the OID of the queried schema.

  Return type: oid

  Example:

```
openGauss=# SELECT get_schema_oid('public');
 get_schema_oid
----------------
           2200
(1 row)
```

- get_client_info()

  Description: Returns client information.

  Return type: record

Access Permission Query Functions

The DDL permissions, including ALTER, DROP, COMMENT, INDEX and VACUUM, are inherent permissions implicitly owned by the owner.

The following access permission query function only indicates whether a user has a certain permission on an object. That is, the permission on the object recorded in the **acl** column of the system catalog is returned.

- has_any_column_privilege(user, table, privilege)

  Description: Queries whether a specified user has permission for any column of table.

Table 11-37 Parameter type description

Parameter	Valid Input Parameter Type
user	name, oid
table	text, oid
privilege	text

Return type: Boolean

- has_any_column_privilege(table, privilege)

  Description: Queries whether the current user has permission to access any column of table. For details about the valid parameter types, see **Table 11-37**.

  Return type: Boolean

  has_any_column_privilege checks whether a user can access any column of a table in a particular way. Its parameter possibilities are analogous to **has_table_privilege**, except that the desired access permission must be some combination of SELECT, INSERT, UPDATE, COMMENT or REFERENCES.

  NOTE

  Note that having any of these permissions at the table level implicitly grants it for each column of the table, so **has_any_column_privilege** will always return **true** if **has_table_privilege** does for the same parameters. But **has_any_column_privilege** also succeeds if there is a column-level grant of the permission for at least one column.

- has_column_privilege(user, table, column, privilege)

  Description: Specifies whether a specified user has permission for columns.

Table 11-38 Parameter type description

Parameter	Valid Input Parameter Type
user	name, oid
table	text, oid
column	text, smallint
privilege	text

Return type: Boolean

- has_column_privilege(table, column, privilege)

  Description: Specifies whether the current user has permission to access columns. For details about the valid parameter types, see **Table 11-38**.

  Return type: Boolean

  has_column_privilege checks whether a user can access a column in a particular way. Its argument possibilities are analogous to **has_table_privilege**, with the addition that the column can be specified either by name or attribute number. The desired access permission must be some combination of SELECT, INSERT, UPDATE, COMMENT or REFERENCES.

NOTE

Note that having any of these permissions at the table level implicitly grants it for each column of the table.

- has_cek_privilege(user, cek, privilege)

  Description: Specifies whether a specified user has permission for accessing CEKs. The parameters are described as follows:

Table 11-39 Parameter type description

Parameter	Valid Input Parameter Type	Description	Range
user	name, oid	User	User name or ID
cek	text, oid	Column encryption key	Name or ID of a CEK.
privilege	text	Permission	• **USAGE**: allows users to use the specified CEK. • **DROP**: allows users to delete the specified CEK.

  Return type: Boolean

- has_cmk_privilege(user, cmk, privilege)

  Description: Specifies whether a specified user has permission for accessing CMKs. The parameters are described as follows:

Table 11-40 Parameter type description

Parameter	Valid Input Parameter Type	Description	Range
user	name, oid	User	User name or ID
cmk	text, oid	CMK	Name or ID of the CMK
privilege	text	Permission	• **USAGE**: allows users to use the specified CMK. • **DROP**: allows users to delete the specified CMK.

Return type: Boolean

- has_database_privilege(user, database, privilege)

  Description: Specifies whether a specified user has permission for accessing databases. The parameters are described as follows:

Table 11-41 Parameter type description

Parameter	Valid Input Parameter Type
user	name, oid
database	text, oid
privilege	text

Return type: Boolean

- has_database_privilege(database, privilege)

  Description: Specifies whether the current user has permission to access a database. For details about the valid parameter types, see **Table 11-41**.

  Return type: Boolean

  Note: **has_database_privilege** checks whether a user can access a database in a particular way. Its argument possibilities are analogous to **has_table_privilege**. The desired access permission must be some combination of CREATE, CONNECT, TEMPORARY, ALTER, DROP, COMMENT or TEMP (which is equivalent to TEMPORARY).

- has_directory_privilege(user, directory, privilege)

  Description: Specifies whether a specified user has permission for accessing directories.

Table 11-42 Parameter type description

Parameter	Valid Input Parameter Type
user	name, oid
directory	text, oid
privilege	text

Return type: Boolean

- has_directory_privilege(directory, privilege)

  Description: Specifies whether the current user has permission to access a directory. For details about the valid parameter types, see **Table 11-42**.

  Return type: Boolean

- has_foreign_data_wrapper_privilege(user, fdw, privilege)

  Description: Specifies whether a specified user has permission for accessing foreign-data wrappers.

Table 11-43 Parameter type description

Parameter	Valid Input Parameter Type
user	name, oid
fdw	text, oid
privilege	text

Return type: Boolean

- has_foreign_data_wrapper_privilege(fdw, privilege)

  Description: Specifies whether the current user has permission for accessing foreign-data wrappers. For details about the valid parameter types, see **Table 11-43**.

  Return type: Boolean

  Note: **has_foreign_data_wrapper_privilege** checks whether a user can access a foreign-data wrapper in a particular way. Its argument possibilities are analogous to **has_table_privilege**. The desired access permission must be USAGE.

- has_function_privilege(user, function, privilege)

  Description: Specifies whether a specified user has permission for accessing functions.

Table 11-44 Parameter type description

Parameter	Valid Input Parameter Type
user	name, oid
function	text, oid
privilege	text

Return type: Boolean

- has_function_privilege(function, privilege)

  Description: Specifies whether the current user has permission for accessing functions. For details about the valid parameter types, see **Table 11-44**.

  Return type: Boolean

  Note: **has_function_privilege** checks whether a user can access a function in a particular way. Its argument possibilities are analogous to **has_table_privilege**. When a function is specified by a text string rather than by OID, the allowed input is the same as that for the **regprocedure** data type (see **OID Types**). The access permission must be EXECUTE, ALTER, DROP, or COMMENT.

- has_language_privilege(user, language, privilege)

  Description: Specifies whether a specified user has permission for accessing languages.

Table 11-45 Parameter type description

Parameter	Valid Input Parameter Type
user	name, oid
language	text, oid
privilege	text

Return type: Boolean

- has_language_privilege(language, privilege)

  Description: Specifies whether the current user has permission for accessing languages. For details about the valid parameter types, see **Table 11-45**.

  Return type: Boolean

  Note: **has_language_privilege** checks whether a user can access a procedural language in a particular way. Its argument possibilities are analogous to **has_table_privilege**. The desired access permission must be USAGE.

- has_nodegroup_privilege(user, nodegroup, privilege)

  Description: Checks whether a user has permission to access a database node.

  Return type: Boolean

Table 11-46 Parameter type description

Parameter	Valid Input Parameter Type
user	name, oid
nodegroup	text, oid
privilege	text

- has_nodegroup_privilege(nodegroup, privilege)

  Description: Checks whether a user has permission to access a database node. The parameter is similar to **has_table_privilege**. The access permission must be USAGE, CREATE, COMPUTE, ALTER, or DROP.

  Return type: Boolean

- has_schema_privilege(user, schema, privilege)

  Description: Specifies whether a specified user has permission for accessing schemas.

  Return type: Boolean

- has_schema_privilege(schema, privilege)

  Description: Specifies whether the current user has permission for accessing schemas.

  Return type: Boolean

  Note: **has_schema_privilege** checks whether a user can access a schema in a particular way. Its argument possibilities are analogous to

has_table_privilege. The desired access permission must be some combination of CREATE, USAGE, ALTER, DROP or COMMENT.

- has_server_privilege(user, server, privilege)

  Description: Specifies whether a specified user has permission for accessing foreign servers.

  Return type: Boolean

- has_server_privilege(server, privilege)

  Description: Specifies whether the current user has permission for accessing foreign servers.

  Return type: Boolean

  Note: **has_server_privilege** checks whether a user can access a foreign server in a particular way. Its argument possibilities are analogous to **has_table_privilege**. The access permission must be USAGE, ALTER, DROP, or COMMENT.

- has_table_privilege(user, table, privilege)

  Description: Specifies whether a specified user has permission for accessing tables.

  Return type: Boolean

- has_table_privilege(table, privilege)

  Description: Specifies whether the current user has permission for accessing tables.

  Return type: Boolean

  has_table_privilege checks whether a user can access a table in a particular way. The user can be specified by name, by OID (**pg_authid.oid**), **public** to indicate the PUBLIC pseudo-role, or if the argument is omitted **current_user** is assumed. The table can be specified by name or by OID. When it is specified by name, the name can be schema-qualified if necessary. The desired access permission is specified by a text string, which must be SELECT, INSERT, UPDATE, DELETE, TRUNCATE, REFERENCES, TRIGGER, ALTER, DROP, COMMENT, INDEX or VACUUM. Optionally, **WITH GRANT OPTION** can be added to a permission type to test whether the permission is held with grant option. Also, multiple permission types can be listed separated by commas, in which case the result will be **true** if any of the listed permissions is held.

  Example:

```
openGauss=# SELECT has_table_privilege('tpcds.web_site', 'select');
 has_table_privilege
---------------------
 t
(1 row)

openGauss=# SELECT has_table_privilege('omm', 'tpcds.web_site', 'select,INSERT WITH GRANT
OPTION ');
 has_table_privilege
---------------------
 t
(1 row)
```

- has_tablespace_privilege(user, tablespace, privilege)

  Description: Specifies whether a specified user has permission for tablespaces.

  Return type: Boolean

- has_tablespace_privilege(tablespace, privilege)

  Description: Specifies whether the current user has permission for tablespaces.

  Return type: Boolean

  Note: **has_tablespace_privilege** checks whether a user can access a tablespace in a particular way. Its argument possibilities are analogous to **has_table_privilege**. The access permission must be CREATE, ALTER, DROP, or COMMENT.

- pg_has_role(user, role, privilege)

  Description: Specifies whether a specified user has permission for accessing roles.

  Return type: Boolean

- pg_has_role(role, privilege)

  Description: Specifies whether the current user has permission for accessing roles.

  Return type: Boolean

  Note: **pg_has_role** checks whether a user can access a role in a particular way. Its argument possibilities are analogous to **has_table_privilege**, except that **public** is not allowed as a user name. The desired access permission must be some combination of MEMBER and USAGE. MEMBER denotes direct or indirect membership in the role (that is, the SET ROLE permission), while USAGE denotes the permissions of the role are available without SET ROLE.

- has_any_privilege(user, privilege)

  Description: Queries whether a specified user has certain ANY permission. If multiple permissions are queried at the same time, **true** is returned as long as one permission is obtained.

  Return type: Boolean

Table 11-47 Parameter type description

Parameter	Valid Input Parameter Type	Description	Range
user	name	User	An existing user name.

Parameter	Valid Input Parameter Type	Description	Range

privilege	text	ANY permission	Available values:
			CREATE ANY TABLE [WITH ADMIN OPTION]
			ALTER ANY TABLE [WITH ADMIN OPTION]
			DROP ANY TABLE [WITH ADMIN OPTION]
			SELECT ANY TABLE [WITH ADMIN OPTION]
			INSERT ANY TABLE [WITH ADMIN OPTION]
			UPDATE ANY TABLE [WITH ADMIN OPTION]
			DELETE ANY TABLE [WITH ADMIN OPTION]
			CREATE ANY SEQUENCE [WITH ADMIN OPTION]
			CREATE ANY INDEX [WITH ADMIN OPTION]
			CREATE ANY FUNCTION [WITH ADMIN OPTION]
			EXECUTE ANY FUNCTION [WITH ADMIN OPTION]
			CREATE ANY PACKAGE [WITH ADMIN OPTION]
			EXECUTE ANY PACKAGE [WITH ADMIN OPTION]
			CREATE ANY TYPE [WITH ADMIN OPTION]

Schema Visibility Inquiry Functions

Each function performs the visibility check for one type of database object. For functions and operators, an object in the search path is visible if there is no object of the same name and parameter data type earlier in the path. For operator classes, both name and associated index access method are considered.

All these functions require object OIDs to identify the object to be checked. If you want to test an object by name, it is convenient to use the OID alias types (**regclass**, **regtype**, **regprocedure**, **regoperator**, **regconfig**, or **regdictionary**).

For example, a table is said to be visible if its containing schema is in the search path and no table of the same name appears earlier in the search path. This is equivalent to the statement that the table can be referenced by name without explicit schema qualification. For example, to list the names of all visible tables:

```
openGauss=# SELECT relname FROM pg_class WHERE pg_table_is_visible(oid);
```

- pg_collation_is_visible(collation_oid)

  Description: Specifies whether the collation is visible in search path.

  Return type: Boolean

- pg_conversion_is_visible(conversion_oid)

  Description: Specifies whether the conversion is visible in search path.

  Return type: Boolean

- pg_function_is_visible(function_oid)

  Description: Specifies whether the function is visible in search path.

  Return type: Boolean

- pg_opclass_is_visible(opclass_oid)

  Description: Specifies whether the operator class is visible in search path.

  Return type: Boolean

- pg_operator_is_visible(operator_oid)

  Description: Specifies whether the operator is visible in search path.

  Return type: Boolean

- pg_opfamily_is_visible(opclass_oid)

  Description: Specifies whether the operator family is visible in search path.

  Return type: Boolean

- pg_table_is_visible(table_oid)

  Description: Specifies whether the table is visible in search path.

  Return type: Boolean

- pg_ts_config_is_visible(config_oid)

  Description: Specifies whether the text search configuration is visible in search path.

  Return type: Boolean

- pg_ts_dict_is_visible(dict_oid)

  Description: Specifies whether the text search dictionary is visible in search path.

  Return type: Boolean

- pg_ts_parser_is_visible(parser_oid)

  Description: Specifies whether the text search parser is visible in search path.

  Return type: Boolean

- pg_ts_template_is_visible(template_oid)

  Description: Specifies whether the text search template is visible in search path.

  Return type: Boolean

- pg_type_is_visible(type_oid)

  Description: Specifies whether the type (or domain) is visible in search path.

  Return type: Boolean

System Catalog Information Functions

- format_type(type_oid, typemod)

  Description: Obtains the SQL name of a data type.

  Return type: text

  Note: **format_type** returns the SQL name of a data type that is identified by its type OID and possibly a type modifier. Pass NULL for the type modifier if no specific modifier is known. Certain type modifiers are passed for data types with length limitations. The SQL name returned from **format_type** contains the length of the data type, which can be calculated by taking sizeof(int32) from actual storage length [actual storage len - sizeof(int32)] in the unit of bytes. 32-bit space is required to store the customized length set by users. So the actual storage length contains 4 bytes more than the customized length. In the following example, the SQL name returned from **format_type** is character varying(6), indicating the length of varchar type is 6 bytes. So the actual storage length of varchar type is 10 bytes.

  ```
  openGauss=# SELECT format_type((SELECT oid FROM pg_type WHERE typname='varchar'), 10);
       format_type
  ----------------------
   character varying(6)
  (1 row)
  ```

- getdistributekey(table_name)

  Description: Obtains a distribution column for a hash table. Distribution is not supported in a standalone system and the return value of this function is empty.

- pg_check_authid(role_oid)

  Description: Checks whether a role name with a given OID exists.

  Return type: Boolean

  Example:

  ```
  openGauss=# select pg_check_authid(1);
  pg_check_authid
  -----------------
  f
  (1 row)
  ```

- pg_describe_object(catalog_id, object_id, object_sub_id)

  Description: Obtains the description of a database object.

  Return type: text

  Note: **pg_describe_object** returns a description of a database object specified by catalog OID, object OID and a (possibly zero) sub-object ID. This is useful to determine the identity of an object as stored in the **pg_depend** catalog.

- pg_get_constraintdef(constraint_oid)

  Description: Obtains the definition of a constraint.

  Return type: text

- pg_get_constraintdef(constraint_oid, pretty_bool)

  Description: Obtains the definition of a constraint.

  Return type: text

  Note: **pg_get_constraintdef** and **pg_get_indexdef** respectively reconstruct the creating command for a constraint and an index.

- pg_get_expr(pg_node_tree, relation_oid)

  Description: Decompiles internal form of an expression, assuming that any Vars in it refer to the relationship indicated by the second parameter.

  Return type: text

- pg_get_expr(pg_node_tree, relation_oid, pretty_bool)

  Description: Decompiles internal form of an expression, assuming that any Vars in it refer to the relationship indicated by the second parameter.

  Return type: text

  Note: **pg_get_expr** decompiles the internal form of an individual expression, such as the default value for a column. It can be useful when examining the contents of system catalogs. If the expression might contain Vars, specify the OID of the relationship they refer to as the second parameter; if no Vars are expected, zero is sufficient.

- pg_get_functiondef(func_oid)

  Description: Obtains the definition of a function.

  Return type: text

  Example:

```
openGauss=# select * from pg_get_functiondef(598);
 headerlines |                      definition
-------------+----------------------------------------------------
           4 | CREATE OR REPLACE FUNCTION pg_catalog.abbrev(inet)+
             |  RETURNS text                                     +
             |  LANGUAGE internal                                +
             |  IMMUTABLE STRICT NOT FENCED NOT SHIPPABLE        +
             | AS $function$inet_abbrev$function$                +
             |
(1 row)
```

- pg_get_function_arguments(func_oid)

  Description: Obtains the parameter list of the function's definition (with default values).

  Return type: text

  Note: **pg_get_function_arguments** returns the parameter list of a function, in the form it would need to appear in within **CREATE FUNCTION**.

- pg_get_function_identity_arguments(func_oid)

  Description: Obtains the parameter list to identify a function (without default values).

  Return type: text

  Note: **pg_get_function_identity_arguments** returns the parameter list necessary to identify a function, in the form it would need to appear in within **ALTER FUNCTION**. This form omits default values.

- pg_get_function_result(func_oid)

  Description: Obtains the **RETURNS** clause for a function.

  Return type: text

  Note: **pg_get_function_result** returns the appropriate **RETURNS** clause for the function.

- pg_get_indexdef(index_oid)

  Description: Obtains the **CREATE INDEX** command for an index.

  Return type: text

Example:

```
openGauss=# select * from pg_get_indexdef(16416);
                     pg_get_indexdef
-------------------------------------------------------------------------
 CREATE INDEX test3_b_idx ON test3 USING btree (b) TABLESPACE pg_default
(1 row)
```

- pg_get_indexdef(index_oid, dump_schema_only)

  Description: Obtains the **CREATE INDEX** command for indexes in dump scenarios. For an interval partitioned table that contains a local index, if **dump_schema_only** is set to **true**, the returned index creation statement does not contain the local index information of the automatically created partition. If **dump_schema_only** is set to **false**, the returned index creation statement contains the local index information of the automatically created partition. For a non-interval partitioned table or an interval partitioned table that does not contain a local index, the value of **dump_schema_only** does not affect the returned result of the function.

  Return type: text

  Example:

```
openGauss=# CREATE TABLE sales
openGauss-# (prod_id NUMBER(6),
openGauss(#  cust_id NUMBER,
openGauss(#  time_id DATE,
openGauss(#  channel_id CHAR(1),
openGauss(#  promo_id NUMBER(6),
openGauss(#  quantity_sold NUMBER(3),
openGauss(#  amount_sold NUMBER(10,2)
openGauss(# )
PARTITION BY RANGE( time_id) INTERVAL('1 day')
openGauss-# (
openGauss(#  partition p1 VALUES LESS THAN ('2019-02-01 00:00:00'),
openGauss(#  partition p2 VALUES LESS THAN ('2019-02-02 00:00:00')
openGauss(# );
CREATE TABLE
openGauss=# create index index_sales on sales(prod_id) local (PARTITION idx_p1 ,PARTITION idx_p2);
CREATE INDEX
openGauss=#-- If the data to be inserted does not match any partition, create a partition and insert
the data into the new partition.
openGauss=# INSERT INTO sales VALUES(1, 12, '2019-02-05 00:00:00', 'a', 1, 1, 1);
INSERT 0 1
openGauss=# select oid from pg_class where relname = 'index_sales';
  oid
-------
 24632
(1 row)
openGauss=# select * from pg_get_indexdef(24632, true);
                                    pg_get_indexdef
--------------------------------------------------------------------------------------------------------------------------
---
 CREATE INDEX index_sales ON sales USING btree (prod_id) LOCAL(PARTITION idx_p1, PARTITION
idx_p2)  TABLESPACE pg_default
(1 row)
openGauss=# select * from pg_get_indexdef(24632, false);
                                              pg_get_indexdef

--------------------------------------------------------------------------------------------------------------------------
-------------
--------------------
 CREATE INDEX index_sales ON sales USING btree (prod_id) LOCAL(PARTITION idx_p1, PARTITION
idx_p2, PARTITION sys_p1_prod_id_idx)  TA
BLESPACE pg_default
(1 row
```

- pg_get_indexdef(index_oid, column_no, pretty_bool)

Description: Obtains the **CREATE INDEX** command for an index, or definition of just one index column when **column_no** is not zero.

Example:

```
openGauss=# select * from pg_get_indexdef(16416, 0, false);
                        pg_get_indexdef
-------------------------------------------------------------------------
 CREATE INDEX test3_b_idx ON test3 USING btree (b) TABLESPACE pg_default
(1 row)
openGauss=# select * from pg_get_indexdef(16416, 1, false);
 pg_get_indexdef
-----------------
 b
(1 row)
```

Return type: text

Note: **pg_get_functiondef** returns a complete **CREATE OR REPLACE FUNCTION** statement for a function.

- pg_get_keywords()

  Description: Obtains the list of SQL keywords and their categories.

  Return type: SETOF record

  Note: **pg_get_keywords** returns a set of records describing the SQL keywords recognized by the server. The **word** column contains the keyword. The **catcode** column contains a category code: **U** for unreserved, **C** for column name, **T** for type or function name, or **R** for reserved. The **catdesc** column contains a possibly-localized string describing the category.

- pg_get_userbyid(role_oid)

  Description: Obtains the role name with a given OID.

  Return type: name

  Note: **pg_get_userbyid** extracts a role's name given its OID.

- pg_check_authid(role_id)

  Description: Checks whether a user exists based on **role_id**.

  Return type: text

  Example:

```
openGauss=# select pg_check_authid(20);
pg_check_authid
-----------------
f
(1 row)
```

- pg_get_viewdef(view_name)

  Description: Obtains the underlying **SELECT** command for a view.

  Return type: text

- pg_get_viewdef(view_name, pretty_bool)

  Description: Obtains the underlying **SELECT** command for a view, lines with columns are wrapped to 80 columns if **pretty_bool** is set to **true**.

  Return type: text

  Note: **pg_get_viewdef** reconstructs the **SELECT** query that defines a view. Most of these functions come in two variants. When the function has the parameter **pretty_bool** and the value is true, it can optionally "pretty-print" the result. The pretty-printed format is more readable. The other one is default format which is more likely to be interpreted the same way by future versions of PostgreSQL. Avoid using pretty-printed output for dump purposes.

Passing **false** for the pretty-print parameter yields the same result as the variant that does not have the parameter at all.

- pg_get_viewdef(view_oid)

  Description: Obtains the underlying **SELECT** command for a view.

  Return type: text

- pg_get_viewdef(view_oid, pretty_bool)

  Description: Obtains the underlying **SELECT** command for a view, lines with columns are wrapped to 80 columns if **pretty_bool** is set to **true**.

  Return type: text

- pg_get_viewdef(view_oid, wrap_column_int)

  Description: Obtains the underlying **SELECT** command for a view, wrapping lines with columns as specified, printing is implied.

  Return type: text

- pg_get_tabledef(table_oid)

  Description: Obtains a table definition based on **table_oid**.

  Example:

```
openGauss=# select * from pg_get_tabledef(16384);
                pg_get_tabledef
-------------------------------------------------------
 SET search_path = public;                            +
 CREATE  TABLE t1 (                                   +
         c1 bigint DEFAULT nextval('serial'::regclass)+
 )                                                    +
 WITH (orientation=row, compression=no)               +
 TO GROUP group1;
(1 row)
```

  Return type: text

- pg_get_tabledef(table_name)

  Description: Obtains a table definition based on **table_name**.

  Example:

```
openGauss=# select * from pg_get_tabledef('t1');
                pg_get_tabledef
-------------------------------------------------------
 SET search_path = public;                            +
 CREATE  TABLE t1 (                                   +
         c1 bigint DEFAULT nextval('serial'::regclass)+
 )                                                    +
 WITH (orientation=row, compression=no)               +
 TO GROUP group1;
(1 row)
```

  Return type: text

  Remarks: **pg_get_tabledef** reconstructs the **CREATE** statement of the table definition, including the table definition, index information, and comments. Users need to create the dependent objects of the table, such as groups, schemas, tablespaces, and servers. The table definition does not include the statements for creating these dependent objects.

- pg_options_to_table(reloptions)

  Description: Obtains the set of storage option name/value pairs.

  Return type: SETOF record

  Note: **pg_options_to_table** returns the set of storage option name/value pairs (**option_name**/**option_value**) when passed **pg_class.reloptions** or **pg_attribute.attoptions**.

- pg_tablespace_databases(tablespace_oid)

  Description: Obtains the set of database OIDs that have objects in the specified tablespace.

  Return type: SETOF oid

  Note: **pg_tablespace_databases** allows a tablespace to be examined. It returns the set of OIDs of databases that have objects stored in the tablespace. If this function returns any rows, the tablespace is not empty and cannot be dropped. To display the specific objects populating the tablespace, you will need to connect to the databases identified by **pg_tablespace_databases** and query their **pg_class** catalogs.

- pg_tablespace_location(tablespace_oid)

  Description: Obtains the path in the file system that this tablespace is located in.

  Return type: text

- pg_typeof(any)

  Description: Obtains the data type of any value.

  Return type: regtype

  Note: **pg_typeof** returns the OID of the data type of the value that is passed to it. This can be helpful for troubleshooting or dynamically constructing SQL queries. The function is declared as returning **regtype**, which is an OID alias type (see **OID Types**). This means that it is the same as an OID for comparison purposes but displays as a type name.

  Example:

```
openGauss=# SELECT pg_typeof(33);
 pg_typeof
-----------
 integer
(1 row)

openGauss=# SELECT typlen FROM pg_type WHERE oid = pg_typeof(33);
 typlen
--------
      4
(1 row)
```

- collation for (any)

  Description: Obtains the collation of the parameter.

  Return type: text

  Note: The expression **collation for** returns the collation of the value that is passed to it.

  Example:

```
openGauss=# SELECT collation for (description) FROM pg_description LIMIT 1;
 pg_collation_for
------------------
 "default"
(1 row)
```

  The value might be quoted and schema-qualified. If no collation is derived for the argument expression, then a null value is returned. If the parameter is not of a collectable data type, then an error is thrown.

- pg_extension_update_paths(name)

  Description: Returns the version update path of the specified extension.

Return type: text(source text), text(path text), text(target text)

- pg_get_serial_sequence(tablename, colname)

  Description: Obtains the sequence of the corresponding table name and column name.

  Return type: text

  Example:
```
openGauss=# select * from pg_get_serial_sequence('t1', 'c1');
 pg_get_serial_sequence
------------------------
 public.serial
(1 row)
```

- pg_sequence_parameters(sequence_oid)

  Description: Obtains the parameters of a specified sequence, including the start value, minimum value, maximum value, and incremental value.

  Return type: int16, int16, int16, int16, Boolean

  Example:
```
openGauss=# select * from pg_sequence_parameters(16420);
 start_value | minimum_value |    maximum_value    | increment | cycle_option
-------------+---------------+---------------------+-----------+--------------
         101 |             1 | 9223372036854775807 |         1 | f
(1 row)
```

Comment Information Functions

- col_description(table_oid, column_number)

  Description: Obtains the comment for a table column.

  Return type: text

  Note: **col_description** returns the comment for a table column, which is specified by the OID of its table and its column number.

- obj_description(object_oid, catalog_name)

  Description: Obtains the comment for a database object.

  Return type: text

  Note: The two-parameter form of **obj_description** returns the comment for a database object specified by its OID and the name of the containing system catalog. For example, **obj_description(123456,'pg_class')** would retrieve the comment for the table with OID 123456. The one-parameter form of **obj_description** requires only the object OID.

  obj_description cannot be used for table columns since columns do not have OIDs of their own.

- obj_description(object_oid)

  Description: Obtains the comment for a database object.

  Return type: text

- shobj_description(object_oid, catalog_name)

  Description: Obtains the comment for a shared database object.

  Return type: text

  Note: **shobj_description** is used just like **obj_description** except the former is used for retrieving comments on shared objects. Some system catalogs are global to all databases in GaussDB, and the comments for objects in them are stored globally as well.

Transaction IDs and Snapshots

Internal transaction IDs (XIDs) are 64 bits. **txid_snapshot**, the data type used by these functions, stores information about transaction ID visibility at a particular moment. **Table 11-48** describes its components.

Table 11-48 Snapshot components

Name	Description
xmin	Earliest transaction ID (txid) that is still active. All earlier transactions will either be committed and visible, or rolled back.
xmax	First as-yet-unassigned txid. All txids greater than or equal to this are not yet started as of the time of the snapshot, so they are invisible.
xip_list	Active txids at the time of the snapshot. The list includes only those active txids between **xmin** and **xmax**; there might be active txids higher than **xmax**. A txid that is **xmin <= txid < xmax** and not in this list was already completed at the time of the snapshot, and is either visible or dead according to its commit status. The list does not include txids of subtransactions.

txid_snapshot's textual representation is **xmin:xmax:xip_list**.

For example, **10:20:10,14,15** means **xmin=10, xmax=20, xip_list=10, 14, 15**.

The following functions provide server transaction information in an exportable form. The main use of these functions is to determine which transactions were committed between two snapshots.

- txid_current()

  Description: Obtains the current transaction ID.

  Return type: bigint

- gs_txid_oldestxmin()

  Description: Obtains the minimum transaction ID (specified by **oldesxmin**).

  Return type: bigint

- txid_current_snapshot()

  Description: Obtains the current snapshot.

  Return type: txid_snapshot

- txid_snapshot_xip(txid_snapshot)

  Description: Obtains in-progress transaction IDs in a snapshot.

  Return type: SETOF bigint

- txid_snapshot_xmax(txid_snapshot)

  Description: Obtains **xmax** of snapshots.

  Return type: bigint

- txid_snapshot_xmin(txid_snapshot)

  Description: Obtains **xmin** of snapshots.

Return type: bigint

- txid_visible_in_snapshot(bigint, txid_snapshot)

  Description: Specifies whether the transaction ID is visible in a snapshot (do not use subtransaction IDs).

  Return type: Boolean

- get_local_prepared_xact()

  Description: Obtains the two-phase residual transaction information of the current node, including the transaction ID, GID of the two-phase transaction, prepared time, owner OID, database OID, and node name of the current node.

  Return type: xid, text, timestamptz, oid, oid, text

- get_remote_prepared_xacts()

  Description: Obtains the two-phase residual transaction information of all remote nodes, including the transaction ID, GID of the two-phase transaction, prepared time, owner name, database name, and node name.

  Return type: xid, text, timestamptz, name, name, text

- global_clean_prepared_xacts(text, text)

  Description: Concurrently cleans two-phase residual transactions. Only the **gs_clean** tool can call this function for the cleaning. In other situations, **false** is returned.

  Return type: Boolean

- gs_get_next_xid_csn()

  Description: Returns the values of **next_xid** and **next_csn** on all nodes globally.

  The return values are as follows:

Table 11-49 gs_get_next_xid_csn parameters

Column	Description
nodename	Node name.
next_xid	Next transaction ID of the current node.
next_csn	Next CSN of the current node.

- slice(hstore, text[])

  Description: Extracts the subset of the hstore type.

  Return type: hstore

  Example:

```
openGauss=# select slice('a=>1,b=>2,c=>3'::hstore, ARRAY['b','c','x']);
      slice
--------------------
 "b"=>"2", "c"=>"3"
(1 row)
```

- slice_array(hstore, text[])

  Description: Extracts the set of hstore values.

  Return type: value array

Example:

```
openGauss=# select slice_array('a=>1,b=>2,c=>3'::hstore, ARRAY['b','c','x']);
 slice_array
-------------
 {2,3,NULL}
(1 row)
```

- skeys(hstore)

  Description: Returns a set of all keys of the hstore type.

  Return type: a set of keys

  Example:

  ```
  openGauss=# select skeys('a=>1,b=>2');
   skeys
  -------
   a
   b
  (2 rows)
  ```

- pg_control_system()

  Description: Returns the status of the system control file.

  Return type: SETOF record

- pg_control_checkpoint()

  Description: Returns the system checkpoint status.

  Return type: SETOF record

- pv_builtin_functions

  Description: Displays information about all built-in system functions.

  Parameter: nan

  Return type: proname name, pronamespace oid, proowner oid, prolang oid, procost real, prorows real, provariadic oid, protransform regproc, proisagg boolean, proiswindow boolean, prosecdef boolean, proleakproof boolean, proisstrict boolean, proretset boolean, provolatile "char", pronargs smallint, pronargdefaults smallint, prorettype oid, proargtypes oidvector, proallargtypes integer[], proargmodes "char"[], proargnames text[], proargdefaults pg_node_tree, prosrc text, probin text, proconfig text[], proacl aclitem[], prodefaultargpos int2vector, fencedmode boolean, proshippable boolean, propackage boolean, oid oid

- pv_thread_memory_detail

  Description: Returns the memory information of each thread.

  Parameter: nan

  Return type: threadid text, tid bigint, thrdtype text, contextname text, level smallint, parent text, totalsize bigint, freesize bigint, usedsize bigint

- pg_relation_compression_ratio

  Description: Queries the compression rate of a table. By default, **1.0** is returned.

  Parameter: text

  Return type: real

- pg_relation_with_compression

  Description: Specifies whether a table is compressed.

  Parameter: text

Return type: Boolean

- pg_stat_file_recursive

  Description: Lists all files in a path.

  Parameter: location text

- pg_shared_memory_detail

  Description: Returns usage information about all generated shared memory contexts. For details about each column, see **GS_SHARED_MEMORY_DETAIL**.

  Parameter: nan

  Return type: contextname text, level smallint, parent text, totalsize bigint, freesize bigint, usedsize bigint

- get_gtm_lite_status

  Description: Returns the backup XID and CSN on the GTM for fault locating. This system function is not supported in GTM-FREE mode or centralized deployment.

- gs_stat_get_wlm_plan_operator_info

  Description: Obtains operator plan information from the internal hash table.

  Parameter: oid

  Return type: datname text, queryid int8, plan_node_id int4, startup_time int8, total_time int8, actual_rows int8, max_peak_memory int4, query_dop int4, parent_node_id int4, left_child_id int4, right_child_id int4, operation text, orientation text, strategy text, options text, condition text, projection text

- pg_stat_get_partition_tuples_hot_updated

  Description: Returns statistics on the number of hot-updated tuples in a partition with a specified partition ID.

  Parameter: oid

  Return type: bigint

- gs_session_memory_detail_tp

  Description: Returns the memory usage of the session. For details, see **gs_session_memory_detail**.

  Parameter: nan

  Return type: sessid text, sesstype text, contextname text, level smallint, parent text, totalsize bigint, freesize bigint, usedsize bigint

- gs_thread_memory_detail

  Description: Returns the memory information of each thread.

  Parameter: nan

  Return type: threadid text, tid bigint, thrdtype text, contextname text, level smallint, parent text, totalsize bigint, freesize bigint, usedsize bigint

- pg_stat_get_wlm_realtime_operator_info

  Description: Obtains the operator information of the real-time execution plan from the internal hash table.

  Parameter: nan

  Return type: queryid bigint, pid bigint, plan_node_id integer, plan_node_name text, start_time timestamp with time zone, duration bigint, status text, query_dop integer, estimated_rows bigint, tuple_processed bigint, min_peak_memory integer, max_peak_memory integer, average_peak_memory integer, memory_skew_percent integer, min_spill_size integer, max_spill_size integer, average_spill_size integer, spill_skew_percent integer, min_cpu_time bigint, max_cpu_time bigint, total_cpu_time bigint, cpu_skew_percent integer, warning text

- pg_stat_get_wlm_realtime_ec_operator_info

  Description: Obtains the operator information of the EC execution plan from the internal hash table.

  Parameter: nan

  Return type: queryid bigint, plan_node_id integer, plan_node_name text, start_time timestamp with time zone, ec_operator integer, ec_status text, ec_execute_datanode text, ec_dsn text, ec_username text, ec_query text, ec_libodbc_type text, ec_fetch_count bigint

- pg_stat_get_wlm_operator_info

  Description: Obtains the operator information of the execution plan from the internal hash table.

  Parameter: nan

  Return type: queryid bigint, pid bigint, plan_node_id integer, plan_node_name text, start_time timestamp with time zone, duration bigint, query_dop integer, estimated_rows bigint, tuple_processed bigint, min_peak_memory integer, max_peak_memory integer, average_peak_memory integer, memory_skew_percent integer, min_spill_size integer, max_spill_size integer, average_spill_size integer, spill_skew_percent integer, min_cpu_time bigint, max_cpu_time bigint, total_cpu_time bigint, cpu_skew_percent integer, warning text

- pg_stat_get_wlm_node_resource_info

  Description: Obtains the resource information of the current node.

  Parameter: nan

  Return type: min_mem_util integer, max_mem_util integer, min_cpu_util integer, max_cpu_util integer, min_io_util integer, max_io_util integer, used_mem_rate integer

- pg_stat_get_session_wlmstat

  Description: Returns the load information of the current session.

  Parameter: pid integer

  Return type: datid oid, threadid bigint, sessionid bigint, threadpid integer, usesysid oid, appname text, query text, priority bigint, block_time bigint, elapsed_time bigint, total_cpu_time bigint, skew_percent integer, statement_mem integer, active_points integer, dop_value integer, current_cgroup text, current_status text, enqueue_state text, attribute text, is_plana boolean, node_group text, srespool name

- pg_stat_get_wlm_ec_operator_info

  Description: Obtains the operator information of the EC execution plan from the internal hash table.

  Parameter: nan

  Return type: queryid bigint, plan_node_id integer, plan_node_name text, start_time timestamp with time zone, duration bigint, tuple_processed bigint, min_peak_memory integer, max_peak_memory integer, average_peak_memory integer, ec_operator integer, ec_status text, ec_execute_datanode text, ec_dsn text, ec_username text, ec_query text, ec_libodbc_type text, ec_fetch_count bigint

- pg_stat_get_wlm_instance_info

  Description: Returns the load information of the current instance.

  Parameter: nan

  Return type: instancename text, timestamp, timestamp with time zone, used_cpu integer, free_memory integer, used_memory integer, io_await double precision, io_util double precision, disk_read double precision, disk_write double precision, process_read bigint, process_write bigint, logical_read bigint, logical_write bigint, read_counts bigint, write_counts bigint

- pg_stat_get_wlm_instance_info_with_cleanup

  Description: Returns the load information of the current instance and saves the information to the system catalog.

  Parameter: nan

  Return type: instancename text, timestamp, timestamp with time zone, used_cpu integer, free_memory integer, used_memory integer, io_await double precision, io_util double precision, disk_read double precision, disk_write double precision, process_read bigint, process_write bigint, logical_read bigint, logical_write bigint, read_counts bigint, write_counts bigint

- pg_stat_get_wlm_realtime_session_info

  Description: Returns the load information of the real-time session.

  Parameter: nan

  Return type: nodename text, threadid bigint, block_time bigint, duration bigint, estimate_total_time bigint, estimate_left_time bigint, schemaname text, query_band text, spill_info text, control_group text, estimate_memory integer, min_peak_memory integer, max_peak_memory integer, average_peak_memory integer, memory_skew_percent integer, min_spill_size integer, max_spill_size integer, average_spill_size integer, spill_skew_percent integer, min_dn_time bigint, max_dn_time bigint, average_dn_time bigint, dntime_skew_percent integer, min_cpu_time bigint, max_cpu_time bigint, total_cpu_time bigint, cpu_skew_percent integer, min_peak_iops integer, max_peak_iops integer, average_peak_iops integer, iops_skew_percent integer, warning text, query text, query_plan text, cpu_top1_node_name text, cpu_top2_node_name text, cpu_top3_node_name text, cpu_top4_node_name text, cpu_top5_node_name text, mem_top1_node_name text, mem_top2_node_name text, mem_top3_node_name text, mem_top4_node_name text, mem_top5_node_name text, cpu_top1_value bigint, cpu_top2_value bigint, cpu_top3_value bigint, cpu_top4_value bigint, cpu_top5_value bigint, mem_top1_value bigint, mem_top2_value bigint, mem_top3_value bigint, mem_top4_value bigint, mem_top5_value bigint, top_mem_dn text, top_cpu_dn text

- pg_stat_get_wlm_session_iostat_info

  Description: Returns the session load I/O information.

  Parameter: nan

  Return type: threadid bigint, maxcurr_iops integer, mincurr_iops integer, maxpeak_iops integer, minpeak_iops integer, iops_limits integer, io_priority integer, curr_io_limits integer

- pg_stat_get_wlm_statistics

  Description: Returns session load statistics.

  Parameter: nan

  Return type: statement text, block_time bigint, elapsed_time bigint, total_cpu_time bigint, qualification_time bigint, skew_percent integer, control_group text, status text, action text

11.5.26 System Administration Functions

11.5.26.1 Configuration Settings Functions

Configuration setting functions are used for querying and modifying configuration parameters during running.

- current_setting(setting_name)

  Description: Specifies the current setting.

  Return type: text

  Note: **current_setting** obtains the current setting of **setting_name** by query. It is equivalent to the **SHOW** statement.

  Example:

```
openGauss=# SELECT current_setting('datestyle');

 current_setting
-----------------
 ISO, MDY
(1 row)
```

- set_working_grand_version_num_manually(tmp_version)

  Description: Upgrades new features of the database by switching the authorization version.

  Return type: void

- shell_in(type)

  Description: Inputs a route for the shell type that has not yet been filled.

  Return type: void

- shell_out(type)

  Description: Outputs a route for the shell type that has not yet been filled.

  Return type: void

- set_config(setting_name, new_value, is_local)

  Description: Sets the parameter and returns a new value.

  Return type: text

  Note: **set_config** sets **setting_name** to **new_value**. If **is_local** is set to **true**, **new_value** applies only to the current transaction. If you want **new_value** to apply for the current session, set the value to **false** instead. The function corresponds to the **SET** statement.

  Example:

```
openGauss=# SELECT set_config('log_statement_stats', 'off', false);

 set_config
------------
```

```
 off
(1 row)
```

11.5.26.2 Universal File Access Functions

Universal file access functions provide local access interfaces for files on a database server. Only files in the database directory and the **log_directory** directory can be accessed. Use a relative path for files in the database directory, and a path matching the **log_directory** configuration setting for log files. Only database initialization users can use these functions.

- pg_ls_dir(dirname text)

  Description: Lists files in a directory.

  Return type: setof text

  Note: **pg_ls_dir** returns all the names in the specified directory, except the special entries "." and "..".

  Example:

```
openGauss=# SELECT pg_ls_dir('./');
      pg_ls_dir
----------------------
 .postgresql.conf.swp
 postgresql.conf
 pg_tblspc
 PG_VERSION
 pg_ident.conf
 core
 server.crt
 pg_serial
 pg_twophase
 postgresql.conf.lock
 pg_stat_tmp
 pg_notify
 pg_subtrans
 pg_ctl.lock
 pg_xlog
 pg_clog
 base
 pg_snapshots
 postmaster.opts
 postmaster.pid
 server.key.rand
 server.key.cipher
 pg_multixact
 pg_errorinfo
 server.key
 pg_hba.conf
 pg_replslot
 .pg_hba.conf.swp
 cacert.pem
 pg_hba.conf.lock
 global
 gaussdb.state
(32 rows)
```

- pg_read_file(filename text, offset bigint, length bigint)

  Description: Returns the content of a text file.

  Return type: text

  Note: **pg_read_file** returns part of a text file. It can return a maximum of *length* bytes from *offset*. The actual size of fetched data is less than *length* if the end of the file is reached first. If *offset* is negative, it is the length rolled back from the file end. If *offset* and *length* are omitted, the entire file is returned.

Example:

```
openGauss=# SELECT pg_read_file('postmaster.pid',0,100);
          pg_read_file
---------------------------------------
 53078                               +
 /srv/BigData/hadoop/data1/dbnode+
 1500022474                          +
 8000                               +
 /var/run/FusionInsight              +
 localhost                           +
  2
(1 row)
```

- pg_read_binary_file(filename text [, offset bigint, length bigint,missing_ok boolean])

  Description: Returns the content of a binary file.

  Return type: bytea

  Note: **pg_read_binary_file** is similar to **pg_read_file**, except that the result is a **bytea** value; accordingly, no encoding checks are performed. In combination with the **convert_from** function, this function can be used to read a file in a specified encoding.

```
openGauss=# SELECT convert_from(pg_read_binary_file('filename'), 'UTF8');
```

- pg_stat_file(filename text)

  Description: Returns status information about a file.

  Return type: record

  Note: **pg_stat_file** returns a record containing the file size, last access timestamp, last modification timestamp, last file status change timestamp, and a Boolean value indicating if it is a directory. Typical use cases are as follows:

```
openGauss=# SELECT * FROM pg_stat_file('filename');
openGauss=# SELECT (pg_stat_file('filename')).modification;
```

  Example:

```
openGauss=# SELECT convert_from(pg_read_binary_file('postmaster.pid'), 'UTF8');
           convert_from
--------------------------------------
 4881                               +
 /srv/BigData/gaussdb/data1/dbnode+
 1496308688                         +
 25108                              +
 /opt/user/Bigdata/gaussdb/gaussdb_tmp +
 *                                 +
  25108001  43352069                +

(1 row)
openGauss=# SELECT * FROM pg_stat_file('postmaster.pid');

 size |         access          |      modification      |         change
| creation | isdir
------+------------------------+------------------------+------------------------
+----------+-------
  117 | 2017-06-05 11:06:34+08 | 2017-06-01 17:18:08+08 | 2017-06-01 17:18:08+08
|          | f
(1 row)
openGauss=# SELECT (pg_stat_file('postmaster.pid')).modification;
      modification
------------------------
 2017-06-01 17:18:08+08
(1 row)
```

11.5.26.3 Server Signal Functions

Server signal functions send control signals to other server processes. Only the system administrator can use these functions.

- pg_cancel_backend(pid int)

  Description: Cancels the current query of a backend.

  Return type: Boolean

  Note: **pg_cancel_backend** sends a query cancellation (SIGINT) signal to the backend process identified by **pid**. The PID of an active backend process can be found in the **pid** column of the **pg_stat_activity** view, or can be found by listing the database process using **ps** on the server. A user with the **SYSADMIN** permission, the owner of the database connected to the backend process, the owner of the backend process, or a user who inherits the **gs_role_signal_backend** permission of the built-in role has the permission to use this function.

- pg_cancel_session(pid bigint, sessionid bigint)

  Description: Cancels a backend session.

  Return type: Boolean

  Note: The input parameters of **pg_cancel_session** can be queried using the **pid** and **sessionid** fields in **pg_stat_activity**. It can be used to clear inactive sessions in thread pool mode.

- pg_reload_conf()

  Description: Causes all server processes to reload their configuration files (restricted to the system administrator).

  Return type: Boolean

  Note: **pg_reload_conf** sends a SIGHUP signal to the server. As a result, all server processes reload their configuration files.

- pg_rotate_logfile()

  Description: Rotates the log files of the server (restricted to the system administrator).

  Return type: Boolean

  Note: **pg_rotate_logfile** sends a signal to the log file manager, instructing the manager to immediately switch to a new output file. This function works only when **redirect_stderr** is used for log output. Otherwise, no log file manager subprocess exists.

- pg_terminate_backend(pid int)

  Description: Terminates a backend thread.

  Return type: Boolean

  Note: Each of these functions returns **true** if they are successful and **false** otherwise. A user with the **SYSADMIN** permission, the owner of the database connected to the backend process, the owner of the backend process, or a user who inherits the **gs_role_signal_backend** permission of the built-in role has the permission to use this function.

  Example:

```
openGauss=# SELECT pid from pg_stat_activity;
      pid
-----------------
```

```
 140657876268816
(1 rows)

openGauss=# SELECT pg_terminate_backend(140657876268816);
 pg_terminate_backend
----------------------
 t
(1 row)
```

- pg_terminate_session(pid int64, sessionid int64)

  Description: Terminates a backend session.

  Return type: Boolean

  Note: Each of these functions returns **true** if they are successful and **false** otherwise. A user with the **SYSADMIN** permission, the owner of the database connected to the session, the owner of the session, or a user who inherits the **gs_role_signal_backend** permission of the built-in role has the permission to use this function.

11.5.26.4 Backup and Restoration Control Functions

Backup Control Functions

Backup control functions help with online backup.

- pg_create_restore_point(name text)

  Description: Creates a named point for performing the restoration operation (restricted to the system administrator).

  Return type: text

  Note: **pg_create_restore_point** creates a named transaction log record that can be used as a restoration target, and returns the corresponding transaction log location. The given name can then be used with **recovery_target_name** to specify the point up to which restoration will proceed. Avoid creating multiple restoration points with the same name, since restoration will stop at the first one whose name matches the restoration target.

- pg_current_xlog_location()

  Description: Obtains the write position of the current transaction log.

  Return type: text

  Note: **pg_current_xlog_location** displays the write position of the current transaction log in the same format as those of the previous functions. Read-only operations do not require permissions of the system administrator.

- pg_current_xlog_insert_location()

  Description: Obtains the insert position of the current transaction log.

  Return type: text

  Note: **pg_current_xlog_insert_location** displays the insert position of the current transaction log. The insertion point is the logical end of the transaction log at any instant, while the write location is the end of what has been written out from the server's internal buffers. The write position is the end that can be detected externally from the server. This operation can be performed to archive only some of completed transaction log files. The insert position is mainly used for commissioning the server. Read-only operations do not require permissions of the system administrator.

- gs_current_xlog_insert_end_location()

  Description: Obtains the insert position of the current transaction log.

  Return type: text

  Note: **gs_current_xlog_insert_end_location** displays the insert position of the current transaction log.

- pg_start_backup(label text [, fast boolean])

  Description: Starts executing online backup (restricted to the system administrator or replication roles).

  Return type: text

  Note: **pg_start_backup** receives a user-defined backup label (usually the name of the position where the backup dump file is stored). This function writes a backup label file to the data directory of the database and then returns the start position of backed up transaction logs in text mode.

```
openGauss=# SELECT pg_start_backup('label_goes_here');
 pg_start_backup
-----------------
 0/3000020
(1 row)
```

- pg_stop_backup()

  Description: Completes online backup (restricted to the system administrator or replication roles).

  Return type: text

  Note: **pg_stop_backup** deletes the label file created by **pg_start_backup** and creates a backup history file in the transaction log archive area. The history file includes the label given to **pg_start_backup**, the start and end transaction log locations for the backup, and the start and end time of the backup. The return value is the backup's ending transaction log location. After the end position is calculated, the insert position of the current transaction log automatically goes ahead to the next transaction log file. In this way, the ended transaction log file can be immediately archived so that backup is complete.

- pg_switch_xlog()

  Description: Switches to a new transaction log file (restricted to the system administrator).

  Return type: text

  Note: **pg_switch_xlog** moves to the next transaction log file so that the current log file can be archived (if continuous archive is used). The return value is the ending transaction log location + 1 within the just-completed transaction log file. If there has been no transaction log activity since the last transaction log switchover, **pg_switch_xlog** will do nothing but return the start location of the transaction log file currently in use.

- pg_xlogfile_name(location text)

  Description: Converts the position string in a transaction log to a file name.

  Return type: text

  Note: **pg_xlogfile_name** extracts only the transaction log file name. If the given transaction log position is the transaction log file border, a transaction log file name will be returned for both the two functions. This is usually the desired behavior for managing transaction log archiving, since the preceding file is the last one that currently needs to be archived.

- pg_xlogfile_name_offset(location text)

  Description: Converts the position string in a transaction log to a file name and returns the byte offset in the file.

  Return type: text and integer

  Note: **pg_xlogfile_name_offset** can extract transaction log file names and byte offsets from the returned results of the preceding functions. Example:

```
openGauss=# SELECT * FROM pg_xlogfile_name_offset(pg_stop_backup());
NOTICE:  pg_stop_backup cleanup done, waiting for required WAL segments to be archived
NOTICE:  pg_stop_backup complete, all required WAL segments have been archived
        file_name         | file_offset
--------------------------+-------------
000000010000000000000003  |         272
(1 row)
```

- pg_xlog_location_diff(location text, location text)

  Description: Calculates the difference in bytes between two transaction log locations.

  Return type: numeric

- pg_cbm_tracked_location()

  Description: Queries the LSN location parsed by CBM.

  Return type: text

- pg_cbm_get_merged_file(startLSNArg text, endLSNArg text)

  Description: Combines CBM files within the specified LSN range into one and returns the name of the combined file.

  Return type: text

  Note: Only the system administrator or O&M administrator can obtain the CBM combination file.

- pg_cbm_get_changed_block(startLSNArg text, endLSNArg text)

  Description: Combines CBM files within the specified LSN range into a table and return records of this table.

  Return type: record

  Note: The table columns include the start LSN, end LSN, tablespace OID, database OID, table relfilenode, table fork number, whether the table is deleted, whether the table is created, whether the table is truncated, number of pages in the truncated table, number of modified pages, and list of modified page numbers.

- pg_cbm_recycle_file(targetLSNArg text)

  Description: Deletes the CBM files that are no longer used and returns the first LSN after the deletion.

  Return type: text

- pg_cbm_force_track(targetLSNArg text,timeOut int)

  Description: Forcibly executes the CBM trace to the specified Xlog position and returns the Xlog position of the actual trace end point.

  Return type: text

- pg_enable_delay_ddl_recycle()

  Description: Enables DDL delay and returns the Xlog position of the enabling point. You need to enable **operation_mode** as the administrator or O&M administrator.

Return type: text

- pg_disable_delay_ddl_recycle(barrierLSNArg text, isForce bool)

  Description: Disables DDL delay and returns the Xlog range where DDL delay takes effect. You need to enable **operation_mode** as the administrator or O&M administrator.

  Return type: record

- pg_enable_delay_xlog_recycle()

  Description: Enables Xlog recycle delay. This function is used in primary database node restoration.

  Return type: void

- pg_disable_delay_xlog_recycle()

  Description: Disables Xlog recycle delay. This function is used in primary database node restoration.

  Return type: void

- pg_cbm_rotate_file(rotate_lsn text)

  Description: Forcibly switches the file after the CBM parses **rotate_lsn**. This function is called during the build process.

  Return type: void

- gs_roach_stop_backup(backupid text)

  Description: Stops a backup started by the internal backup tool GaussRoach. It is similar to the **pg_stop_backup system** function but is more lightweight.

  Return type: text. The content is the insertion position of the current log.

- gs_roach_enable_delay_ddl_recycle(backupid name)

  Description: Enables DDL delay and returns the log position of the enabling point. It is similar to the **pg_enable_delay_ddl_recycle** system function but is more lightweight. In addition, different **backupid** values can be used to concurrently open DDL statements with delay.

  Return type: text. The content is the log location of the start point.

- gs_roach_disable_delay_ddl_recycle(backupid text)

  Description: Disables DDL delay, returns the range of logs on which DDL delay takes effect, and deletes the physical files of column-store tables that are deleted by users within this range. It is similar to the **pg_enable_delay_ddl_recycle** system function but is more lightweight. In addition, the DDL delay function can be disabled concurrently by specifying different backupid values.

  Return type: record. The content is the range of logs for which DDL is delayed to take effect.

- gs_roach_switch_xlog(request_ckpt bool)

  Description: Switches the currently used log segment file and triggers a full checkpoint if **request_ckpt** is set to **true**.

  Return type: text. The content is the location of the segment log.

- gs_block_dw_io(timeout int, identifier text)

  Description: Blocks doublewrite page flushing.

  Parameter description:

  - timeout

    Block duration.

    Value range: [0,3600] (s). The value **0** indicates that the block duration is 0s.

  - identifier

    ID of the operation.

    Value range: a string, supporting only uppercase letters, lowercase letters, digits, and underscores (_).

  Return type: Boolean

  Note: To call this function, the user must have the SYSADMIN or OPRADMIN permission, and **operate_mode** must be enabled for the O&M administrator role.

- gs_is_dw_io_blocked()

  Description: Checks whether disk flushing on the current doublewrite page is blocked. If disk flushing is blocked, **true** is returned.

  Return type: Boolean

  Note: To call this function, the user must have the SYSADMIN or OPRADMIN permission, and **operate_mode** must be enabled for the O&M administrator role.

Restoration Control Functions

Restoration control functions provide information about the status of standby nodes. These functions may be executed both during restoration and in normal running.

- pg_is_in_recovery()

  Description: Returns **true** if restoration is still in progress.

  Return type: Boolean

- pg_last_xlog_receive_location()

  Description: Obtains the last transaction log location received and synchronized to disk by streaming replication. While streaming replication is in progress, this will increase monotonically. If restoration has been completed, then this value will remain static at the value of the last WAL record received and synchronized to disk during restoration. If streaming replication is disabled or if it has not yet started, the function returns **NULL**.

  Return type: text

- pg_last_xlog_replay_location()

  Description: Obtains last transaction log location replayed during restoration. If restoration is still in progress, this will increase monotonically. If restoration has been completed, then this value will remain static at the value of the last WAL record received during that restoration. When the server has been started normally without restoration, the function returns **NULL**.

  Return type: text

- pg_last_xact_replay_timestamp()

  Description: Obtains the timestamp of last transaction replayed during restoration. This is the time to commit a transaction or abort a WAL record on the primary node. If no transactions have been replayed during restoration, this function will return **NULL**. If restoration is still in progress, this will increase monotonically. If restoration has been completed, then this value will remain static at the value of the last WAL record received during that restoration. If the server normally starts without manual intervention, this function will return **NULL**.

Return type: timestamp with time zone

Restoration control functions control restoration processes. These functions may be executed only during restoration.

- pg_is_xlog_replay_paused()

  Description: Returns **true** if restoration is paused.

  Return type: Boolean
- pg_xlog_replay_pause()

  Description: Pauses restoration immediately.

  Return type: void
- pg_xlog_replay_resume()

  Description: Restarts restoration if it was paused.

  Return type: void
- gs_get_active_archiving_standby()

  Description: Queries information about archive standby nodes in the same shard. The standby node name, archive location, and number of archived logs are returned.

  Return type: text, text, int
- gs_pitr_get_warning_for_xlog_force_recycle()

  Description: Checks whether logs are recycled because a large number of logs are stacked in the archive slot after archiving is enabled.

  Return type: Boolean
- gs_pitr_clean_history_global_barriers(stop_barrier_timestamp cstring)

  Description: Clears all barrier records generated before the specified time. The earliest barrier record is returned. The input parameter is of the cstring type and is a Linux timestamp. You need to perform this operation as the administrator or O&M administrator.

  Return type: text
- gs_pitr_archive_slot_force_advance(stop_barrier_timestamp cstring)

  Description: Forcibly pushes the archive slot and clears unnecessary barrier records. The new archive slot location is returned. The input parameter is of the cstring type and is a Linux timestamp. You need to perform this operation as the administrator or O&M administrator.

  Return type: text

While restoration is paused, no further database changes are applied. In hot standby mode, all new queries will see the same consistent snapshot of the database, and no further query conflicts will be generated until restoration is resumed.

If streaming replication is disabled, the paused state may continue indefinitely without problem. While streaming replication is in progress, WAL records will continue to be received, which will eventually fill available disk space. This progress depends on the duration of the pause, the rate of WAL generation, and available disk space.

11.5.26.5 DR Control Functions for Dual Database Instances

DR control functions for dual-database instances can be used to create an archive slot. The archive slot specifies the OBS information for storing physical logs.

- pg_create_physical_replication_slot_extern(slotname text, dummy_standby bool, extra_content text, need_recycle_xlog bool)

  Description: Creates an OBS or a NAS archive slot. **slotname** indicates the slot name of the DR standby node. The primary and standby nodes must use the same slot name. **dummy_standby** specifies whether the database is deployed in primary/standby/secondary mode or one-primary and multi-standby mode. The value **false** indicates that the database is deployed in one-primary and multi-standby mode, and the value **true** indicates that the database is deployed in primary/standby/secondary mode. **extra_content** contains some information about the archive slot. For an OBS archive slot, the format is **OBS;obs_server_ip;obs_bucket_name;obs_ak;obs_sk;archive_path;is_recovery;is_vote_replicate**, where **OBS** indicates the archive media of the archive slot, **obs_server_ip** indicates the IP address of OBS, **obs_bucket_name** indicates the bucket name, **obs_ak** indicates the AK of OBS, **obs_sk** indicates the SK of OBS, **archive_path** indicates the archive path **i**, and **is_recovery** indicates whether the slot is an archive slot or a recovery slot. The value **0** indicates that the slot is an archive slot and is used by the primary database instance. The value **1** indicates that the slot is a recovery slot and is used by the DR database instance. **is_vote_replicate** indicates whether the voting copy is archived first. The value **0** indicates that the synchronous standby server is archived first, and the value **1** indicates that the voting copy is archived first. This field is reserved in the current version and is not adapted yet. For a NAS archive slot, the format is **NAS;archive_path;is_recovery;is_vote_replicate**. Compared with the OBS archive slot, the NAS archive slot does not have the OBS configuration information, while the meanings of other fields are the same.

  If the media is not specified, the OBS archive slot is used by default. The **extra_content** format is **obs_server_ip;obs_bucket_name;obs_ak;obs_sk;archive_path;is_recovery;is_vote_replicate**.

- **need_recycle_xlog** specifies whether to recycle old archived logs when creating an archive slot. The value **true** indicates that old archived logs are recycled, and the value **false** indicates that old archive logs are not recycled.

  Return type: records, including **slotname** and **xlog_position** of the current DR standby.

  Note: Users who invoke this function must have the **SYSADMIN** permission or the **REPLICATION** permission, or inherit the **gs_role_replication** permission of the built-in role.

  Example:

  Create an OBS archive slot.

```
openGauss=# select * from pg_create_physical_replication_slot_extern('uuid', false, 'OBS;obs.cn-north-7.ulanqab.huawei.com;dyk;19D772JBCACXX3KWS51D;********;openGauss_uuid/dn1;0;0', false);
 slotname | xlog_position
```

```
----------+---------------
 uuid     |
(1 row)
```

Create a NAS archive slot.

```
openGauss=# select * from pg_create_physical_replication_slot_extern('uuid', false, 'NAS;/data/nas/
media/openGauss_uuid/dn1;0;0',false);
 slotname | xlog_position
----------+---------------
 uuid     |
```

- gs_set_obs_delete_location(delete_location text)

  Description: Sets the location where OBS archived logs can be deleted. The value of **delete_location** is an LSN. The logs before this location have been replayed and flushed to disks in the DR database instance and can be deleted on OBS.

  Return type: xlog_file_name text, indicating the name of the log file that can be deleted. The value of this parameter is returned regardless of whether OBS is deleted successfully.

```
openGauss=# select gs_set_obs_delete_location('0/54000000');
 gs_set_obs_delete_location
-----------------------------
 0000000100000000000000054_00
(1 row)
```

- gs_set_obs_delete_location_with_slotname(cstring, cstring)

  Description: Sets the location where OBS archive logs can be deleted in a DR relationship. The first parameter indicates the LSN. The logs before this location have been replayed and flushed to disks in the DR database instance and can be deleted on OBS. The second parameter indicates the name of the archive slot.

  Return type: xlog_file_name text, indicating the name of the log file that can be deleted. The value of this parameter is returned regardless of whether OBS is deleted successfully.

- gs_hadr_do_switchover()

  Description: Triggers a planned switchover in the primary database instance in OBS-based remote DR solutions.

  Return type: Boolean, indicating whether the switchover process is performed normally and whether services are taken over successfully.

- gs_streaming_dr_in_switchover()

  Description: Triggers a planned switchover in the primary database instance in streaming replication-based remote DR solutions.

  Return type: Boolean, indicating whether the switchover process is performed normally and whether services are taken over successfully.

11.5.26.6 DR Query Functions for Dual-Database Instances

- gs_get_global_barrier_status()

  Description: If the 3DC geo-redundant DR is enabled, logs of the primary database instance and DR database instance are synchronized through OBS. The barrier log is flushed to the disk of the primary database instance. The progress of archiving logs of the primary database instance and the progress of replaying logs of the DR database instance are determined by replaying the DR database instance. **gs_get_global_barrier_status** is used to query the latest global barrier that has been archived in OBS for the primary database instance.

Return type: text

global_barrier_id: globally latest barrier ID.

global_achive_barrier_id: globally latest archived barrier ID.

- gs_get_global_barriers_status()

  Description: If 3DC geo-redundant DR solutions based on OBS are enabled, logs of the primary database instance and multiple DR database instances are synchronized through OBS. The barrier log is flushed to the disk of the primary database instance. The progress of archiving logs of the primary database instance and the progress of replaying logs of the DR database instance are determined by replaying the DR database instances. **gs_get_global_barriers_status** is used to query the latest global barrier that has been archived in OBS for the primary database instance.

  Return type: text

  slot_name: name of the slot used for DR.

  global_barrier_id: globally latest barrier ID.

  global_achive_barrier_id: globally latest archived barrier ID.

- gs_get_local_barrier_status()

  Description: If 3DC geo-redundant DR is enabled, logs of the primary database instance and DR database instance are synchronized through OBS. The barrier log is flushed to the disk of the primary database instance. The progress of archiving logs of the primary database instance and the progress of replaying logs of the DR database instance are determined by replaying the DR database instance. **gs_get_local_barrier_status** is used to query the current log replay status of each node of the DR database instance.

  Return type: text

  barrier_id: latest barrier ID of a node of the DR database instance.

  barrier_lsn: LSN of the latest barrier ID returned by a node in the DR database instance.

  archive_lsn: location of archived logs obtained by a node in the DR database instance. This parameter does not take effect currently.

  flush_lsn: location of logs that have been flushed to disks on a node in the DR database instance.

- gs_upload_obs_file('slot_name', 'src_file', 'dest_file')

  Description: Function used by the primary database instance to upload data to OBS if 3DC geo-redundant DR is enabled.

  Return type: void

  slot_name: name of the replication slot created for the primary database instance.

  src_file: location of the file to be uploaded in the data directory of the primary database instance.

  dest_file: location of the file to be uploaded to OBS.

- gs_download_obs_file('slot_name', 'src_file', 'dest_file')

  Description: Function used by the DR database instance to download data from OBS to the local host if 3DC geo-redundant DR is enabled.

Return type: void

slot_name: name of the replication slot created for the DR database instance.

src_file: location of files to be downloaded from OBS.

dest_file: location of the downloaded file in the data directory of the DR database instance.

- gs_get_obs_file_context('file_name', 'slot_name')

  Description: Queries file content on OBS if 3DC geo-redundant DR is enabled.

  Return type: text

  file_name: name of the file on OBS.

  slot_name: name of the replication slot created for the primary or DR database instance.

- gs_set_obs_file_context('file_name', 'file_context','slot_name')

  Description: Creates a file on OBS and writes content into the file if 3DC geo-redundant DR is enabled.

  Return type: text

  file_name: name of the file on OBS.

  file_context: content to be written into the file.

  slot_name: name of the replication slot created for the primary or DR database instance.

- gs_get_hadr_key_cn()

  Description: Creates a file on OBS and writes content into the file if 3DC geo-redundant DR is enabled.

  Return type: text

  file_name: name of the file on OBS.

  file_context: content to be written into the file.

  slot_name: name of the replication slot created for the primary or DR database instance

- gs_hadr_has_barrier_creator()

  Description: Checks whether the **barrier_creator** thread exists on the current node if 3DC geo-redundant DR is enabled. If yes, **true** is returned (restricted to the system administrator).

  Return type: Boolean

  Note: This function is used only when a planned switchover is performed on the DR database instance.

- gs_hadr_in_recovery()

  Description: Checks whether the current node is in barrier-based log restoration if 3DC geo-redundant DR is enabled. If yes, **true** is returned. The DR database instance can be promoted to the production database instance through the switchover process only after the log restoration is complete (restricted to the system administrator).

  Return type: Boolean

  Note: This function is used only when a planned switchover is performed on the DR database instance.

- gs_streaming_dr_get_switchover_barrier()

Description: Checks whether the DN instance of the DR database instance has received the switchover barrier logs and replayed the logs in 3DC geo-redundant DR solutions based on streaming replication. If it has, **true** is returned. In the DR database instance, the procedure for promoting the DR database instance to the production database instance in the switchover process can be started only after the switchover barrier logs of all DN instances are replayed (restricted to the system administrator).

Return type: Boolean

Note: This function is used only when a planned switchover is performed on the DR database instance in streaming DR solutions.

- gs_streaming_dr_service_truncation_check()

  Description: Checks whether the DN instance of the primary database instance has sent the switchover barrier logs in 3DC geo-redundant DR solutions based on streaming replication. If it has, **true** is returned. The procedure for demoting the production database instance to the DR database instance in the switchover process can be started only after the logs are sent (restricted to the system administrator).

  Return type: Boolean

  Note: This function is used only when a planned switchover is performed on the DR database instance.

  – gs_hadr_local_rto_and_rpo_stat()

  Description: Displays the log flow control information about the primary and standby database instances in the streaming DR scenario. (This view can be used only on the primary DN of the primary database instance. Statistics cannot be obtained from the standby DN or the standby database instance.)

  Return type: record. The types and meanings of the fields are as follows:

Parameter	**Type**	**Description**
hadr_sender_node_name	text	Node name, including the primary database instance and the first standby node of the standby database instance.
hadr_receiver_node_name	text	Name of the first standby node of the standby database instance.
source_ip	text	IP address of the primary DN of the primary database instance.
source_port	int	Communication port of the primary DN of the primary database instance.
dest_ip	text	IP address of the first standby DN of the standby database instance.
dest_port	int	Communication port of the first standby DN of the standby database instance.

Parameter	Type	Description
current_rto	int	Flow control information, that is, log RTO time of the current primary and standby database instances (unit: second).
target_rto	int	Flow control information, that is, the RTO time between the target primary and standby database instances (unit: second).
current_rpo	int	Flow control information, that is, log RPO time of the current primary and standby database instances (unit: second).
target_rpo	int	Flow control information, that is, the RPO time between the target primary and standby database instances (unit: second).
rto_sleep_time	int	RTO flow control information, that is, the expected sleep time (unit: μs) required by walsender on the host to reach the specified RTO.
rpo_sleep_time	int	RPO flow control information, that is, the expected sleep time (unit: μs) required by xlogInsert on the host to reach the specified RPO.

– gs_hadr_remote_rto_and_rpo_stat()

Description: Displays the log flow control information of the primary and standby database instances of other non-local data shards in streaming DR mode. This function is not supported in centralized deployment scenarios.

11.5.26.7 Snapshot Synchronization Functions

Snapshot synchronization functions save the current snapshot and return its identifier.

- pg_export_snapshot()

  Description: Saves the current snapshot and returns its identifier.

  Return type: text

  Note: **pg_export_snapshot** saves the current snapshot and returns a text string identifying the snapshot. This string must be passed to clients that want to import the snapshot. A snapshot can be imported when the **set transaction snapshot snapshot_id;** command is executed. Doing so is possible only when the transaction is set to the **SERIALIZABLE** or **REPEATABLE READ** isolation level. GaussDB does not support these two isolation levels currently. The output of the function cannot be used as the input of **set transaction snapshot**.
- pg_export_snapshot_and_csn()

Description: Saves the current snapshot and returns its identifier. Compared with **pg_export_snapshot()**, **pg_export_snapshot()** returns a CSN, indicating the CSN of the current snapshot.

Return type: text

11.5.26.8 Database Object Functions

Database Object Size Functions

Database object size functions calculate the actual disk space used by database objects.

- pg_column_size(any)

  Description: Specifies the number of bytes used to store a particular value (possibly compressed)

  Return type: int

  Note: **pg_column_size** displays the space for storing an independent data value.

```
openGauss=# SELECT pg_column_size(1);
 pg_column_size
----------------
              4
(1 row)
```

- pg_database_size(oid)

  Description: Specifies the disk space used by the database with the specified OID.

  Return type: bigint

- pg_database_size(name)

  Description: Specifies the disk space used by the database with the specified name.

  Return type: bigint

  Note: **pg_database_size** receives the OID or name of a database and returns the disk space used by the corresponding object.

  Example:

```
openGauss=# SELECT pg_database_size('postgres');
 pg_database_size
------------------
         51590112
(1 row)
```

- pg_relation_size(oid)

  Description: Specifies the disk space used by the table with a specified OID or index.

  Return type: bigint

- get_db_source_datasize()

  Description: Estimates the total size of non-compressed data in the current database.

  Return type: bigint

Remarks: (1) Perform an analysis before this function is called. (2) Calculate the total data capacity in the non-compressed state by estimating the compression rate of the column-store tables.

Example:

```
openGauss=# analyze;
ANALYZE
openGauss=# select get_db_source_datasize();
 get_db_source_datasize
------------------------
            35384925667
(1 row)
```

- pg_relation_size(text)

  Description: Specifies the disk space used by the table with a specified name or index. The table name can be schema-qualified.

  Return type: bigint

- pg_relation_size(relation regclass, fork text)

  Description: Specifies the disk space used by the specified bifurcating tree ('main', 'fsm', or 'vm') of a certain table or index.

  Return type: bigint

- pg_relation_size(relation regclass)

  Description: Is an abbreviation of **pg_relation_size(..., 'main')**.

  Return type: bigint

  Note: **pg_relation_size** receives the OID or name of a table, an index, or a compressed table, and returns the size.

- pg_partition_size(oid,oid)

  Description: Specifies the disk space used by the partition with a specified OID. The first **oid** is the OID of the table and the second **oid** is the OID of the partition.

  Return type: bigint

- pg_partition_size(text, text)

  Description: Specifies the disk space used by the partition with a specified name. The first **text** is the table name and the second **text** is the partition name.

  Return type: bigint

- pg_partition_indexes_size(oid,oid)

  Description: Specifies the disk space used by the index of the partition with a specified OID. The first **oid** is the OID of the table and the second **oid** is the OID of the partition.

  Return type: bigint

- pg_partition_indexes_size(text,text)

  Description: Specifies the disk space used by the index of the partition with a specified name. The first **text** is the table name and the second **text** is the partition name.

  Return type: bigint

- pg_indexes_size(regclass)

  Description: Specifies the total disk space used by the index appended to the specified table.

Return type: bigint

- pg_size_pretty(bigint)

  Description: Converts a size in bytes expressed as a 64-bit integer into a human-readable format with size units.

  Return type: text

- pg_size_pretty(numeric)

  Description: Converts a size in bytes expressed as a numeric value into a human-readable format with size units.

  Return type: text

  Note: **pg_size_pretty** formats the results of other functions into a human-readable format. KB/MB/GB/TB can be used.

- pg_table_size(regclass)

  Description: Specifies the disk space used by the specified table, excluding indexes (but including TOAST, free space mapping, and visibility mapping).

  Return type: bigint

- pg_tablespace_size(oid)

  Description: Specifies the disk space used by the tablespace with a specified OID.

  Return type: bigint

- pg_tablespace_size(name)

  Description: Specifies the disk space used by the tablespace with a specified name.

  Return type: bigint

  Note:

  pg_tablespace_size receives the OID or name of a database and returns the disk space used by the corresponding object.

- pg_total_relation_size(oid)

  Description: Specifies the disk space used by the table with a specified OID, including the index and the compressed data.

  Return type: bigint

- pg_total_relation_size(regclass)

  Description: Specifies the total disk space used by the specified table, including all indexes and TOAST data.

  Return type: bigint

- pg_total_relation_size(text)

  Description: Specifies the disk space used by the table with a specified name, including the index and the compressed data. The table name can be schema-qualified.

  Return type: bigint

  Note: **pg_total_relation_size** receives the OID or name of a table or a compressed table, and returns the sizes of the data, related indexes, and the compressed table in bytes.

- datalength(any)

Description: Specifies the number of bytes used by an expression of a specified data type (data management space, data compression, or data type conversion is not considered).

Return type: int

Note: **datalength** is used to calculate the space of an independent data value.

Example:

```
openGauss=# SELECT datalength(1);
datalength
------------
4
(1 row)
```

The following table lists the supported data types and calculation methods.

Data Type			Storage Space
Numeric data types	Integer types	TINYINT	1
		SMALLINT	2
		INTEGER	4
		BINARY_INTEGER	4
		BIGINT	8
	Arbitrary precision types	DECIMAL	Every four decimal digits occupy two bytes. The digits before and after the decimal point are calculated separately.
		NUMERIC	Every four decimal digits occupy two bytes. The digits before and after the decimal point are calculated separately.
		NUMBER	Every four decimal digits occupy two bytes. The digits before and after the decimal point are calculated separately.
	Sequence integer	SMALLSERIAL	2
		SERIAL	4
		BIGSERIAL	8
		LARGESERIAL	Every four decimal digits occupy two bytes. The digits before and after the decimal point are calculated separately.

	Floating point types	FLOAT4	4
		DOUBLE PRECISION	8
		FLOAT8	8
		BINARY_DOUBLE	8
		FLOAT[(p)]	Every four decimal digits occupy two bytes. The digits before and after the decimal point are calculated separately.
		DEC[(p[,s])]	Every four decimal digits occupy two bytes. The digits before and after the decimal point are calculated separately.
		INTEGER[(p[,s])]	Every four decimal digits occupy two bytes. The digits before and after the decimal point are calculated separately.
Boolean data types	Boolean type	BOOLEAN	1
Character data types	Character types	CHAR	n
		CHAR(n)	n
		CHARACTER(n)	n
		NCHAR(n)	n
		VARCHAR(n)	n
		CHARACTER	Actual number of bytes of a character
		VARYING(n)	Actual number of bytes of a character
		VARCHAR2(n)	Actual number of bytes of a character
		NVARCHAR(n)	Actual number of bytes of a character
		NVARCHAR2(n)	Actual number of bytes of a character
		TEXT	Actual number of bytes of a character

		CLOB	Actual number of bytes of a character
Time data types	Time types	DATE	8
		TIME	8
		TIMEZ	12
		TIMESTAMP	8
		TIMESTAMPZ	8
		SMALLDATETIME	8
		INTERVAL DAY TO SECOND	16
		INTERVAL	16
		RELTIME	4
		ABSTIME	4
		TINTERVAL	12

Database Object Position Functions

- pg_relation_filenode(relation regclass)

  Description: Specifies the ID of a filenode with the specified relationship.

  Return type: oid

  Description: **pg_relation_filenode** receives the OID or name of a table, an index, a sequence, or a compressed table, and returns the number of **filenode** allocated to it. **filenode** is the basic component of the file name used by the relationship. For most tables, the result is the same as that of **pg_class.relfilenode**. For a specified system directory, **relfilenode** is set to **0** and this function must be used to obtain the correct value. If a relationship that is not stored is transmitted, such as a view, this function returns **NULL**.
- pg_relation_filepath(relation regclass)

  Description: Specifies the name of a file path with the specified relationship.

  Return type: text

  Description: **pg_relation_filepath** is similar to **pg_relation_filenode**, except that **pg_relation_filepath** returns the whole file path name for the relationship (relative to the data directory **PGDATA** of the database).
- pg_filenode_relation(tablespace oid, filenode oid)

  Description: Obtains the table names corresponding to the tablespace and relfilenode.

  Return type: regclass
- pg_partition_filenode(partition_oid)

  Description: Obtains **filenode** corresponding to the OID lock of a specified partitioned table.

  Return type: oid

- pg_partition_filepath(partition_oid)

  Description: Specifies the file path name of a partition.

  Return type: text

Recycle Bin Object Functions

- gs_is_recycle_object(classid, objid, objname)

  Description: Determines whether an object is in the recycle bin.

  Return type: Boolean

11.5.26.9 Advisory Lock Functions

Advisory lock functions manage advisory locks.

- pg_advisory_lock(key bigint)

  Description: Obtains an exclusive session-level advisory lock.

  Return type: void

  Note: **pg_advisory_lock** locks resources defined by an application. The resources can be identified using a 64-bit or two nonoverlapped 32-bit key values. If another session locks the resources, the function blocks the resources until they can be used. The lock is exclusive. Multiple locking requests are pushed into the stack. Therefore, if the same resource is locked three times, it must be unlocked three times so that it is released to another session.

- pg_advisory_lock(key1 int, key2 int)

  Description: Obtains an exclusive session-level advisory lock.

  Return type: void

  Note: Only users with the **sysadmin** permission can add session-level exclusive advisory locks to the key-value pair (65535, 65535).

- pg_advisory_lock(int4, int4, Name)

  Description: Obtains the exclusive advisory lock of a specified database.

  Return type: void

- pg_advisory_lock_shared(key bigint)

  Description: Obtains a shared session-level advisory lock.

  Return type: void

- pg_advisory_lock_shared(key1 int, key2 int)

  Description: Obtains a shared session-level advisory lock.

  Return type: void

  Note: **pg_advisory_lock_shared** works in the same way as **pg_advisory_lock**, except the lock can be shared with other sessions requesting shared locks. Only would-be exclusive lockers are locked out.

- pg_advisory_unlock(key bigint)

  Description: Releases an exclusive session-level advisory lock.

  Return type: Boolean

- pg_advisory_unlock(key1 int, key2 int)

  Description: Releases an exclusive session-level advisory lock.

Return type: Boolean

Note: **pg_advisory_unlock** releases the obtained exclusive advisory lock. If the release is successful, the function returns **true**. If the lock was not held, it will return **false**. In addition, a SQL warning will be reported by the server.

- pg_advisory_unlock(int4, int4, Name)

  Description: Releases the exclusive advisory lock of a specified database.

  Return type: Boolean

  Note: If the release is successful, **true** is returned. If no lock is held, **false** is returned.

- pg_advisory_unlock_shared(key bigint)

  Description: Releases a shared session-level advisory lock.

  Return type: Boolean

- pg_advisory_unlock_shared(key1 int, key2 int)

  Description: Releases a shared session-level advisory lock.

  Return type: Boolean

  Note: **pg_advisory_unlock_shared** works in the same way as **pg_advisory_unlock**, except it releases a shared session-level advisory lock.

- pg_advisory_unlock_all()

  Description: Releases all advisory locks owned by the current session.

  Return type: void

  Note: **pg_advisory_unlock_all** releases all advisory locks owned by the current session. The function is implicitly invoked when the session ends even if the client is abnormally disconnected.

- pg_advisory_xact_lock(key bigint)

  Description: Obtains an exclusive transaction-level advisory lock.

  Return type: void

- pg_advisory_xact_lock(key1 int, key2 int)

  Description: Obtains an exclusive transaction-level advisory lock.

  Return type: void

  Note: **pg_advisory_xact_lock** works in the same way as **pg_advisory_lock**, except the lock is automatically released at the end of the current transaction and cannot be released explicitly. Only users with the **sysadmin** permission can add transaction-level exclusive advisory locks to the key-value pair (65535, 65535).

- pg_advisory_xact_lock_shared(key bigint)

  Description: Obtains a shared transaction-level advisory lock.

  Return type: void

- pg_advisory_xact_lock_shared(key1 int, key2 int)

  Description: Obtains a shared transaction-level advisory lock.

  Return type: void

  Note: **pg_advisory_xact_lock_shared** works in the same way as **pg_advisory_lock_shared**, except the lock is automatically released at the end of the current transaction and cannot be released explicitly.

- pg_try_advisory_lock(key bigint)

  Description: Obtains an exclusive session-level advisory lock if available.

  Return type: Boolean

  Note: **pg_try_advisory_lock** is similar to **pg_advisory_lock**, except **pg_try_advisory_lock** does not block the resource until the resource is released. **pg_try_advisory_lock** either immediately obtains the lock and returns **true** or returns **false**, which indicates the lock cannot be performed currently.

- pg_try_advisory_lock(key1 int, key2 int)

  Description: Obtains an exclusive session-level advisory lock if available.

  Return type: Boolean

  Note: Only users with the **sysadmin** permission can add session-level exclusive advisory locks to the key-value pair (65535, 65535).

- pg_try_advisory_lock_shared(key bigint)

  Description: Obtains a shared session-level advisory lock if available.

  Return type: Boolean

- pg_try_advisory_lock_shared(key1 int, key2 int)

  Description: Obtains a shared session-level advisory lock if available.

  Return type: Boolean

  Note: **pg_try_advisory_lock_shared** is similar to **pg_try_advisory_lock**, except **pg_try_advisory_lock_shared** attempts to obtain a shared lock instead of an exclusive lock.

- pg_try_advisory_xact_lock(key bigint)

  Description: Obtains an exclusive transaction-level advisory lock if available.

  Return type: Boolean

- pg_try_advisory_xact_lock(key1 int, key2 int)

  Description: Obtains an exclusive transaction-level advisory lock if available.

  Return type: Boolean

  Note: **pg_try_advisory_xact_lock** works in the same way as **pg_try_advisory_lock**, except the lock, if acquired, is automatically released at the end of the current transaction and cannot be released explicitly. Note: Only users with the **sysadmin** permission can add transaction-level exclusive advisory locks to the key-value pair (65535, 65535).

- pg_try_advisory_xact_lock_shared(key bigint)

  Description: Obtains a shared transaction-level advisory lock if available.

  Return type: Boolean

- pg_try_advisory_xact_lock_shared(key1 int, key2 int)

  Description: Obtains a shared transaction-level advisory lock if available.

  Return type: Boolean

  Note: **pg_try_advisory_xact_lock_shared** works in the same way as **pg_try_advisory_lock_shared**, except the lock, if acquired, is automatically released at the end of the current transaction and cannot be released explicitly.

- lock_cluster_ddl()

Description: Attempts to obtain a session-level exclusive advisory lock for all active primary database nodes in the database.

Return type: Boolean

Note: Only users with the **sysadmin** permission can call this function.

- unlock_cluster_ddl()

  Description: Attempts to add a session-level exclusive advisory lock on the primary database node.

  Return type: Boolean

11.5.26.10 Logical Replication Functions

- pg_create_logical_replication_slot('slot_name', 'plugin_name')

  Description: Creates a logical replication slot.

  Parameter description:

  – slot_name

    Indicates the name of the streaming replication slot.

    Value range: a string of characters, which can contain only lowercase letters, digits, underscores (_), question marks (?), hyphens (-), and periods (.). The '.' or '..' cannot be used as a replication slot name.

  – plugin_name

    Indicates the name of the plugin.

    Value range: a string, supporting **mppdb_decoding**

  Return type: name, text

  Note: The first return value is the slot name, and the second is the start LSN position for decoding in the logical replication slot. Users who invoke this function must have the SYSADMIN permission, the REPLICATION permission, or inherit the gs_role_replication permission of the built-in role. Currently, this function can be invoked only on the host.

- pg_create_physical_replication_slot('slot_name', 'isDummyStandby')

  Description: Creates a physical replication slot.

  Parameter description:

  – slot_name

    Indicates the name of the streaming replication slot.

    Value range: a string of characters, which can contain only lowercase letters, digits, underscores (_), question marks (?), hyphens (-), and periods (.). The '.' or '..' cannot be used as a replication slot name.

  – isDummyStandby

    Specifies whether the replication slot is created by connecting the secondary node to the primary node.

    Type: bool

  Return type: name, text

  Note: Users who invoke this function must have the SYSADMIN permission or the REPLICATION permission, or inherit the gs_role_replication permission of the built-in role. Currently, primary/standby/secondary deployment is not supported by default.

- pg_drop_replication_slot('slot_name')

  Description: Deletes a streaming replication slot.

  Parameter description:

  – slot_name

    Indicates the name of the streaming replication slot.

    Value range: a string of characters, which can contain only lowercase letters, digits, underscores (_), question marks (?), hyphens (-), and periods (.). The '.' or '..' cannot be used as a replication slot name.

  Return type: void

  Note: Users who invoke this function must have the SYSADMIN permission or the REPLICATION permission, or inherit the gs_role_replication permission of the built-in role. Currently, this function can be invoked only on the host.

- pg_logical_slot_peek_changes('slot_name', 'LSN', upto_nchanges, 'options_name', 'options_value')

  Description: Performs decoding but does not go to the next streaming replication slot. (The decoded result will be returned again during the next decoding.)

  Parameter description:

  – slot_name

    Indicates the name of the streaming replication slot.

    Value range: a string of characters, which can contain only lowercase letters, digits, underscores (_), question marks (?), hyphens (-), and periods (.). The '.' or '..' cannot be used as a replication slot name.

  – LSN

    Indicates a target LSN. Decoding is performed only when an LSN is less than or equal to this value.

    Value range: a string, in the format of *xlogid*/*xrecoff*, for example, 1/2AAFC60 (If this parameter is set to **NULL**, the target LSN indicating the end position of decoding is not specified.)

  – upto_nchanges

    Indicates the number of decoded records (including the **begin** and **commit** timestamps). Assume that there are three transactions, which involve 3, 5, and 7 records, respectively. If **upto_nchanges** is set to **4**, 8 records of the first two transactions will be decoded. Specifically, decoding is stopped when the number of decoded records exceeds the value of **upto_nchanges** after decoding in the first two transactions is finished.

    Value range: a non-negative integer

    NOTE

    If any of the **LSN** and **upto_nchanges** values are reached, decoding ends.

  – **options**: Specifies optional parameters, consisting of **options_name** and **options_value**.

    ▪ include-xids

      Indicates whether the decoded **data** column contains XID information.

Valid value: **0** and **1**. The default value is **1**.

- **0**: The decoded **data** column does not contain XID information.
- **1**: The decoded **data** column contains XID information.

- skip-empty-xacts

  Indicates whether to ignore empty transaction information during decoding.

  Valid value: **0** and **1**. The default value is **0**.

  - **0**: The empty transaction information is not ignored during decoding.
  - **1**: The empty transaction information is ignored during decoding.

- include-timestamp

  Indicates whether decoded information contains the **commit** timestamp.

  Valid value: **0** and **1**. The default value is **0**.

  - **0**: The decoded information does not contain the **commit** timestamp.
  - **1**: The decoded information contains the **commit** timestamp.

- only-local

  Specifies whether to decode only local logs.

  Value range: **0** and **1**. The default value is **1**.

  - **0**: Non-local logs and local logs are decoded.
  - **1**: Only local logs are decoded.

- force-binary

  Specifies whether to output the decoding result in binary format.

  Value range: **0**

  - **0**: The decoding result is output in text format.

- white-table-list

  Whitelist parameter, including the schema and table name to be decoded.

  Value range: a string that contains table names in the whitelist. Different tables are separated by commas (,). An asterisk (*) is used to fuzzily match all tables. Schema names and table names are separated by periods (.). No space character is allowed. Example: **select * from pg_logical_slot_peek_changes('slot1', NULL, 4096, 'white-table-list', 'public.t1,public.t2');**

- max-txn-in-memory

  Memory control parameter. The unit is MB. If the memory occupied by a single transaction is greater than the value of this parameter, data is flushed to disks.

  Value range: an integer ranging from 0 to 100. The default value is **0**, indicating that memory control is disabled.

    - max-reorderbuffer-in-memory

      Memory control parameter. The unit is GB. If the total memory (including the cache) of transactions being concatenated in the sender thread is greater than the value of this parameter, the current decoding transaction is flushed to disks.

      Value range: an integer ranging from 0 to 100. The default value is **0**, indicating that memory control is disabled.

  Return type: text, xid, text

  Note: The function returns the decoding result. Each decoding result contains three columns, corresponding to the above return types and indicating the LSN position, XID, and decoded content, respectively.

  Users who invoke this function must have the SYSADMIN permission, the REPLICATION permission, or inherit the gs_role_replication permission of the built-in role.

- pg_logical_slot_get_changes('slot_name', 'LSN', upto_nchanges, 'options_name', 'options_value')

  Description: Performs decoding and goes to the next streaming replication slot.

  Parameter: This function has the same parameters as **pg_logical_slot_peek_changes**. For details, see **pg_logical_slot_peek_ch...**.

  Note: Users who invoke this function must have the SYSADMIN permission or the REPLICATION permission, or inherit the gs_role_replication permission of the built-in role. Currently, this function can be invoked only on the host.

- pg_logical_slot_peek_binary_changes('slot_name', 'LSN', upto_nchanges, 'options_name', 'options_value')

  Description: Performs decoding in binary mode and does not go to the next streaming replication slot. (The decoded data can be obtained again during the next decoding.)

  Parameter description:

  – slot_name

    Indicates the name of the streaming replication slot.

    Value range: a string of characters, which can contain only lowercase letters, digits, underscores (_), question marks (?), hyphens (-), and periods (.). The '.' or '..' cannot be used as a replication slot name.

  – LSN

    Indicates a target LSN. Decoding is performed only when an LSN is less than or equal to this value.

    Value range: a string, in the format of *xlogid/xrecoff*, for example, 1/2AAFC60 (If this parameter is set to **NULL**, the target LSN indicating the end position of decoding is not specified.)

  – upto_nchanges

    Indicates the number of decoded records (including the **begin** and **commit** timestamps). Assume that there are three transactions, which involve 3, 5, and 7 records, respectively. If **upto_nchanges** is set to **4**, 8 records of the first two transactions will be decoded. Specifically, decoding is stopped when the number of decoded records exceeds the value of **upto_nchanges** after decoding in the first two transactions is finished.

Value range: a non-negative integer

NOTE

If any of the **LSN** and **upto_nchanges** values are reached, decoding ends.

– **options**: Specifies optional parameters, consisting of **options_name** and **options_value**.

- include-xids

  Indicates whether the decoded **data** column contains XID information.

  Valid value: **0** and **1**. The default value is **1**.

  - **0**: The decoded **data** column does not contain XID information.
  - **1**: The decoded **data** column contains XID information.

- skip-empty-xacts

  Indicates whether to ignore empty transaction information during decoding.

  Valid value: **0** and **1**. The default value is **0**.

  - **0**: The empty transaction information is not ignored during decoding.
  - **1**: The empty transaction information is ignored during decoding.

- include-timestamp

  Indicates whether decoded information contains the **commit** timestamp.

  Valid value: **0** and **1**. The default value is **0**.

  - **0**: The decoded information does not contain the **commit** timestamp.
  - **1**: The decoded information contains the **commit** timestamp.

- only-local

  Specifies whether to decode only local logs.

  Value range: **0** and **1**. The default value is **1**.

  - **0**: Non-local logs and local logs are decoded.
  - **1**: Only local logs are decoded.

- force-binary

  Specifies whether to output the decoding result in binary format.

  Value range: **0** or **1**. The default value is **0**. The result is output in binary format.

- white-table-list

  Whitelist parameter, including the schema and table name to be decoded.

Value range: a string that contains table names in the whitelist. Different tables are separated by commas (,). An asterisk (*) is used to fuzzily match all tables. Schema names and table names are separated by periods (.). No space character is allowed. Example: **select * from pg_logical_slot_peek_binary_changes('slot1', NULL, 4096, 'white-table-list', 'public.t1,public.t2');**

Return type: text, xid, bytea

Note: The function returns the decoding result. Each decoding result contains three columns, corresponding to the above return types and indicating the LSN position, XID, and decoded content in binary format, respectively. Users who invoke this function must have the SYSADMIN permission, the REPLICATION permission, or inherit the gs_role_replication permission of the built-in role.

- pg_logical_slot_get_binary_changes('slot_name', 'LSN', upto_nchanges, 'options_name', 'options_value')

  Description: Performs decoding in binary mode and does not go to the next streaming replication slot.

  Parameter: This function has the same parameters as **pg_logical_slot_peek_binary_changes**. For details, see **•pg_logical_slot_peek_bi...**

  Note: Users who invoke this function must have the SYSADMIN permission or the REPLICATION permission, or inherit the gs_role_replication permission of the built-in role.

- pg_replication_slot_advance ('slot_name', 'LSN')

  Description: Directly goes to the streaming replication slot for a specified LSN, without outputting any decoded result.

  Parameter description:

  – slot_name

    Indicates the name of the streaming replication slot.

    Value range: a string of characters, which can contain only lowercase letters, digits, underscores (_), question marks (?), hyphens (-), and periods (.). The '.' or '..' cannot be used as a replication slot name.

  – LSN

    Indicates a target LSN. Next decoding will be performed only in transactions whose commission position is greater than this value. If an input LSN is smaller than the position recorded in the current streaming replication slot, the function directly returns. If the input LSN is greater than the LSN of the current physical log, the latter LSN will be directly used for decoding.

    Value range: a string, in the format of *xlogid*/*xrecoff*

  Return type: name, text

  Note: A return result contains the slot name and LSN that is actually used for decoding. Users who invoke this function must have the SYSADMIN permission, the REPLICATION permission, or inherit the gs_role_replication permission of the built-in role. Currently, this function can be invoked only on the host.

- pg_logical_get_area_changes('LSN_start', 'LSN_end', upto_nchanges, 'decoding_plugin', 'xlog_path', 'options_name', 'options_value')

Description: Specifies an LSN range or an Xlog file for decoding when no DDL is available.

The constraints are as follows:

a. When the API is called, only when **wal_level** is set to **logical**, the generated log files can be parsed. If the used Xlog file is not of the logical level, the decoded content does not have the corresponding value and type, and there is no other impact.
b. The Xlog file can be parsed only by a copy of a fully homogeneous DN to ensure that the metadata corresponding to the data can be found and no DDL or VACUUM FULL operation is performed.
c. You can find the XLog to be parsed.
d. Do not read too many Xlog files at a time. You are advised to read one Xlog file at a time. It is estimated that the memory occupied by one Xlog file is two to three times the size of the Xlog file.
e. The Xlog file before scale-out cannot be decoded.

Parameter description:

– LSN_start

  Specifies the LSN at the start of decoding.

  Value range: a string, in the format of *xlogid*/*xrecoff*, for example, 1/2AAFC60 (If this parameter is set to **NULL**, the target LSN indicating the end position of decoding is not specified.)

– LSN_end

  Specifies the LSN at the end of decoding.

  Value range: a string, in the format of *xlogid*/*xrecoff*, for example, 1/2AAFC60 (If this parameter is set to **NULL**, the target LSN indicating the end position of decoding is not specified.)

– upto_nchanges

  Indicates the number of decoded records (including the **begin** and **commit** timestamps). Assume that there are three transactions, which involve 3, 5, and 7 records, respectively. If **upto_nchanges** is set to **4**, 8 records of the first two transactions will be decoded. Specifically, decoding is stopped when the number of decoded records exceeds the value of **upto_nchanges** after decoding in the first two transactions is finished.

  Value range: a non-negative integer

  NOTE

  If any of the **LSN** and **upto_nchanges** values are reached, decoding ends.

– decoding_plugin

Decoding plug-in, which is a .so plug-in that specifies the output format of the decoded content.

Value range: **mppdb_decoding** and **sql_decoding**.

– xlog_path

Decoding plug-in, which specifies the Xlog absolute path and file level of the decoding file.

Value range: **NULL** or a character string of the absolute path of the Xlog file.

– **options**: This parameter is optional and consists of a series of **options_name** and **options_value**. You can retain the default value. For details, see **pg_logical_slot_peek_changes**.

Example:

```
openGauss=# SELECT pg_current_xlog_location();
 pg_current_xlog_location
--------------------------
 0/E62E238
(1 row)

openGauss=# create table t1 (a int primary key,b int,c int);
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "t1_pkey" for table "t1"
CREATE TABLE
openGauss=# insert into t1 values(1,1,1);
INSERT 0 1
openGauss=# insert into t1 values(2,2,2);
INSERT 0 1

openGauss=# select data from pg_logical_get_area_changes('0/
E62E238',NULL,NULL,'sql_decoding',NULL);
 location  |  xid  |                       data
-----------+-------+---------------------------------------------------
 0/E62E8D0 | 27213 | COMMIT (at 2022-01-26 15:08:03.349057+08) 3020226
 0/E6325F0 | 27214 | COMMIT (at 2022-01-26 15:08:07.309869+08) 3020234
......
```

- pg_get_replication_slots()

  Description: Obtains the replication slot list.

  Return type: text, text, text, oid, boolean, xid, xid, text, boolean, text

  Example:

```
openGauss=# select * from pg_get_replication_slots();
 slot_name |     plugin     | slot_type | datoid | active | xmin | catalog_xmin | restart_lsn |
dummy_standby | confirmed_flush
-----------+----------------+-----------+--------+--------+------+--------------+-------------+---------------
+-----------------
 dn_s1     |                | physical  |      0 | t      |      |              | 0/23DB14E0  | f             |
 slot1     | mppdb_decoding | logical   |  16304 | f      |      |        60966 | 0/1AFA1BB0  | f             |
0/23DA5700
(2 rows)
```

- gs_get_parallel_decode_status()

  Description: Monitors the length of the read log queue and decoding result queue of each decoder thread to locate the concurrent decoding performance bottleneck.

  Return type: text, int, text, text, text, int64, int64

  Example:

```
openGauss=# select * from gs_get_parallel_decode_status();
 slot_name | parallel_decode_num | read_change_queue_length  | decode_change_queue_length |
reader_lsn | working_txn_cnt | working_txn_memory
-----------+---------------------+---------------------------+----------------------------+------------
+-----------------+--------------------
 slot1     |                   2 | queue0: 1005, queue1: 320 | queue0: 63, queue1: 748    | 0/1DCE2578
|              42 |          192927504
(1 row)
```

  Note: In the return values, **slot_name** indicates the replication slot name, **parallel_decode_num** indicates the number of parallel decoder threads in the replication slot, **read_change_queue_length** indicates the current length of the log queue read by each decoder thread, **decode_change_queue_length** indicates the current length of the decoding result queue of each decoder thread, **reader_lsn** indicates the log location read by the reader thread,

working_txn_cnt indicates the number of transactions being concatenated in the current sender thread, and **working_txn_memory** indicates the total memory (in bytes) occupied by the concatenation transactions in the sender thread.

- pg_replication_origin_create (node_name)

  Description: Creates a replication source with a given external name and returns the internal ID assigned to it.

  Note: The user who calls this function must have the SYSADMIN permission.

  Parameter description:

  – node_name

    Name of the replication source to be created.

    Value range: a string, supporting only letters, digits, and the following special characters: _?-.

  Return type: oid

- pg_replication_origin_drop (node_name)

  Description: Deletes a previously created replication source, including any associated replay progress.

  Note: The user who calls this function must have the SYSADMIN permission.

  Parameter description:

  – node_name

    Name of the replication source to be deleted.

    Value range: a string, supporting only letters, digits, and the following special characters: _?-.

- pg_replication_origin_oid (node_name)

  Description: Searches for a replication source by name and returns the internal ID. If no such replication source is found, an error is thrown.

  Note: The user who calls this function must have the SYSADMIN permission.

  Parameter description:

  – node_name

    Specifies the name of the replication source to be queried.

    Value range: a string, supporting only letters, digits, and the following special characters: _?-.

  Return type: oid

- pg_replication_origin_session_setup (node_name)

  Description: Marks the current session for replaying from a given origin, allowing you to track replay progress. This parameter can be used only when no origin is selected. Run the **pg_replication_origin_session_reset** command to cancel the configuration.

  Note: The user who calls this function must have the SYSADMIN permission.

  Parameter description:

  – node_name

    Name of the replication source.

    Value range: a string, supporting only letters, digits, and the following special characters: _?-.

- pg_replication_origin_session_reset ()

  Description: Cancels the **pg_replication_origin_session_setup()** effect.

  Note: The user who calls this function must have the SYSADMIN permission.

- pg_replication_origin_session_is_setup ()

  Description: Returns a true value if a replication source is selected in the current session.

  Note: The user who calls this function must have the SYSADMIN permission.

  Return type: Boolean

- pg_replication_origin_session_progress (flush)

  Description: Returns the replay position of the replication source selected in the current session.

  Note: The user who calls this function must have the SYSADMIN permission.

  Parameter description:

  – flush

    Determines whether the corresponding local transaction has been flushed to disk.

    Value range: Boolean

  Return type: LSN

- pg_replication_origin_xact_setup (origin_lsn, origin_timestamp)

  Description: Marks the current transaction as recommitted at a given LSN and timestamp. This function can be called only when **pg_replication_origin_session_setup** is used to select a replication source.

  Note: The user who calls this function must have the SYSADMIN permission.

  Parameter description:

  – origin_lsn

    Position for replaying the replication source.

    Value range: LSN

  – origin_timestamp

    Time point when a transaction is committed

    Value range: timestamp with time zone

- pg_replication_origin_xact_reset ()

  Description: Cancels the **pg_replication_origin_xact_setup()** effect.

  Note: The user who calls this function must have the SYSADMIN permission.

- pg_replication_origin_advance (node_name, lsn)

  Description:

  Sets the replication progress of a given node to a given position. This is primarily used to set the initial position, or to set a new position after a configuration change or similar change.

  Note: Improper use of this function may cause inconsistent replication data.

  Note: The user who calls this function must have the SYSADMIN permission.

  Parameter description:

  – node_name

    Name of an existing replication source.

    Value range: a string, supporting only letters, digits, and the following special characters: _?-.

  - lsn

    Position for replaying the replication source.

    Value range: LSN

- pg_replication_origin_progress (node_name, flush)

  Description: Returns the position for replaying the given replication source.

  Note: The user who calls this function must have the SYSADMIN permission.

  Parameter description:

  - node_name

    Name of the replication source.

    Value range: a string, supporting only letters, digits, and the following special characters: _?-.

  - flush

    Determines whether the corresponding local transaction has been flushed to disk.

    Value range: Boolean

- pg_show_replication_origin_status()

  Description: Displays the replication status of the replication source.

  Note: The user who calls this function must have the SYSADMIN permission.

  Return type:

  - **local_id**: OID, which specifies the ID of the replication source.
  - **external_id**: text, which specifies the name of the replication source.
  - **remote_lsn**: LSN of the replication source.
  - **local_lsn**: local LSN.

- pg_get_publication_tables(pub_name)

  Description: Returns the relid list of tables to be published based on the publication name.

  Parameter description:

  - pub_name

    Existing publication name.

    Value range: a string, supporting only letters, digits, and the following special characters: _?-.

  Return type: relid list

- pg_stat_get_subscription(sub_oid oid) → record

  Description:

  Rerurns the subscription status information after a subscription OID is entered.

  Parameter description:

- subid

  Subscription OID.

Value range: oid

Return type:

- **relid**: OID of the table.
- **pid**: thread_id, which indicates the thread ID of the background apply/sync thread.
- **received_lsn**: pg_lsn, which indicates the latest LSN received from the publisher.
- **last_msg_send_time**: timestamp, which indicates the time when the last message is sent from the publisher.
- **last_msg_receipt_time**: timestamp, which indicates the time when the last message is received by the subscriber.
- **latest_end_lsn**: pg_lsn, which indicates the LSN of the publisher when the last keepalive message is received.
- **latest_end_time**: timstamp, which indicates the time when the last keepalive message is received.

11.5.26.11 Segment-Page Storage Functions

- local_segment_space_info(tablespacename TEXT, databasename TEXT)

  Description: Generates usage information about all extent groups in the tablespace.

  Return type:

node_name	Node name
extent_size	Extent specifications of an extent group. The unit is the number of blocks.
forknum	Fork number
total_blocks	Total number of extents in a physical file
meta_data_blocks	Number of blocks occupied by the metadata managed in a tablespace, including the space header and map page but excluding the segment head
used_data_blocks	Number of extents used for storing data, including the segment head
utilization	Percentage of the number of used blocks to the total number of blocks, that is, (the value of **used_data_blocks** + the value of **meta_data_block**)/the value of **total_blocks**
high_water_mark	High-water mark, indicating the number of allocated extents and maximum physical page number. Blocks that exceed the high-water mark are not used and can be directly recycled.

Example:

```
select * from local_segment_space_info('pg_default', 'postgres');
     node_name      | extent_size | forknum | total_blocks | meta_data_blocks | used_data_blocks | 
utilization | high_water_mark
-------------------+-------------+---------+--------------+------------------+------------------+-------------
+-----------------
 dn_6001_6002_6003 |           1 |       0 |        16384 |             4157 |                1 |     .253784 | 
4158
 dn_6001_6002_6003 |           8 |       0 |        16384 |             4157 |                8 |     .254211 | 
4165
(2 rows)
```

- pg_stat_segment_extent_usage(int4 tablespace oid, int4 database oid, int4 extent_type, int4 forknum)

  Description: Specifies the usage information of each allocated extent in an extent group returned each time. **extent_type** indicates the type of the extent group. The value is an integer ranging from 1 to 5. If the value is not within the range, an error is reported. **forknum** indicates the fork number. The value is an integer ranging from 0 to 4. Currently, only the following values are valid: **0** for data files, **1** for FSM files, and **2** for visibility map files.

  Return type:

Name	Description
start_block	Start physical page number of an extent
extent_size	Size of an extent
usage_type	Usage type of an extent, for example, **segment head** and **data extent**
ower_location	Object location of an extent to which a pointer points. For example, the owner of a data extent is the head of the segment to which the data extent belongs.
special_data	Position of an extent in its owner. The value of this field is related to the usage type. For example, special data of a data extent is the extent ID in the segment to which the data extent belongs.

  The value of **usage_type** is enumerated. The meaning of each value is as follows:

  – **Non-bucket table segment head**: data segment head of a non-hash bucket table
  – **Non-bucket table fork head**: fork segment header of a non-segment-page table
  – **Data extent**: data block

  Example:

```
select * from pg_stat_segment_extent_usage((select oid::int4 from pg_tablespace where 
spcname='pg_default'), (select oid::int4 from pg_database where datname='postgres'), 1, 0);
start_block | extent_size |        usage_type        | ower_location | special_data
-------------+-------------+------------------------+----------------+--------------
```

```
4157 |          1 | Data extent |    4294967295 |           0
4158 |          1 | Data extent |         4157 |           0
```

- local_space_shrink(tablespacename TEXT, databasename TEXT)

  Description: Shrinks specified physical segment-page space on the current node. Only the currently connected database can be shrank.

  Return value: empty

- gs_space_shrink(int4 tablespace, int4 database, int4 extent_type, int4 forknum)

  Description: Works similar to **local_space_shrink**, that is, shrinks specified physical segment-page space. However, the parameters are different. The input parameters are the OIDs of the tablespace and database, and the value of **extent_type** is an integer ranging from 2 to 5. Note: The value **1** of **extent_type** indicates segment-page metadata. Currently, the physical file that contains the metadata cannot be shrunk. This function is used only by tools. You are not advised to use it directly.

  Return value: empty

- pg_stat_remain_segment_info()

  Description: Displays residual extents on the current node due to faults. Residual extents are classified into two types: segments that are allocated but not used and extents that are allocated but not used. The main difference is that a segment contains multiple extents. During reclamation, all extents in the segment need to be recycled.

  Return type:

Name	Description
space_id	Tablespace ID
db_id	Database ID
block_id	Extent ID
type	Extent type. The options are as follows: **ALLOC_SEGMENT**, **DROP_SEGMENT**, and **SHRINK_EXTENT**.

  The values of **type** are described as follows:

  – **ALLOC_SEGMENT**: When a user creates a segment-page table and the segment is just allocated but the transaction of creating a table is not committed, the node is faulty. As a result, the segment is not used after being allocated.

  – **DROP_SEGMENT**: When a user deletes a segment-page table and the transaction is successfully committed, the bit corresponding to the segment page of the table is not reset and a fault, such as power failure, occurs. As a result, the segment is not used or released.

  – **SHRINK_EXTENT**: When a user shrinks a segment-page table and does not release the idle extent, a fault, such as power failure, occurs. As a result, the extent remains and cannot be reused.

    Example:

```
select * from pg_stat_remain_segment_info();
space_id | db_id | block_id | type
----------+-------+----------+------
1663      |   16385|        4156| ALLOC_SEGMENT
```

- pg_free_remain_segment(int4 spaceId, int4 dbId, int4 segmentId)

  Description: Releases a specified residual extent. The value must be obtained from the **pg_stat_remain_segment_info** function. The function verifies input values. If the specified extent is not among the recorded residual extents, an error message is returned. If the specified extent is a single extent, the extent is released independently. If it is a segment, the segment and all extents in the segment are released.

  Return value: empty

11.5.26.12 Other Functions

- plan_seed()

  Description: Obtains the seed value of the previous query statement (internal use).

  Return type: int

- pg_stat_get_env()

  Description: Obtains the environment variable information of the current node. Only users with the sysadmin or monitor admin permission can access the environment variable information.

  Return type: record

  Example:

```
openGauss=# select pg_stat_get_env();
                                                                   pg_stat_get_env
-----------------------------------------------------------------------------------------------------------------------------
----------------
 (sgnode,"localhost,XXX.XXX.XXX.XXX",28589,26000,/home/omm,/home/omm/data/
single_node,pg_log)
(1 row)
```

- pg_catalog.plancache_clean()

  Description: Clears the global plan cache that is not used on nodes.

  Return type: Boolean

- pg_catalog.plancache_status()

  Description: Displays information about the global plan cache on nodes. The information returned by the function is the same as that in **GLOBAL_PLANCACHE_STATUS**.

  Return type: record

- textlen(text)

  Description: Provides the method of querying the logical length of text.

  Return type: int

- threadpool_status()

  Description: Displays the status of worker threads and sessions in the thread pool.

  Return type: record

- get_local_active_session()

Description: Provides sampling records of the historical active sessions stored in the memory of the current node.

Return type: record

- pg_stat_get_thread()

  Description: Provides status information about all threads on the current node. Users with the sysadmin or monitor admin permission can view information about all threads, and common users can view only their own thread information.

  Return type: record

- pg_stat_get_sql_count()

  Description: Provides the counts of the SELECT, UPDATE, INSERT, DELETE, and MERGE INTO statements executed on the current node. Users with the sysadmin or monitor admin permission can view information about all users, and common users can view only their own statistics.

  Return type: record

- pg_stat_get_data_senders()

  Description: Provides detailed information about the data-copy sender thread active at the moment.

  Return type: record

- get_wait_event_info()

  Description: Provides detailed information about the wait event.

  Return type: record

- generate_wdr_report(begin_snap_id bigint, end_snap_id bigint, report_type cstring, report_scope cstring, node_name cstring)

  Description: Generates system diagnosis reports based on two snapshots. You need to run the command in the **postgres** database. By default, the initial user or users with the monadmin permission can access the database. The result can be queried only in the system database but cannot be queried in the user database.

  Return type: record

Table 11-50 generate_wdr_report parameter description

Parameter	Description	Range
begin_snap_id	Snapshot ID that starts the diagnosis report period.	N/A
end_snap_id	Snapshot ID that ends the diagnosis report period. By default, the value of **end_snap_id** is greater than that of **begin_snap_id**.	N/A

report_type	Specifies the type of the generated report.	● **summary** ● **detail** ● **all**: Both **summary** and **detail** types are included.
report_scope	Specifies the scope for a report to be generated.	● **cluster**: database-level information ● **node**: node-level information
node_name	When **report_scope** is set to **node**, set this parameter to the name of the corresponding node. (You can run the **select * from pg_node_env;** command to query the node name.) If **report_scope** is set to **cluster**, this parameter can be omitted, left blank, or set to **NULL**.	● **cluster**: This value is omitted, left blank or set to **NULL**. ● **node**: a node name in GaussDB

- create_wdr_snapshot()

  Description: Manually generates system diagnosis snapshots. This function requires the **sysadmin** permission.

  Return type: text

- kill_snapshot()

  Description: Kills the WDR snapshot backend thread. Users who invoke this function must have the SYSADMIN permission, the REPLICATION permission, or inherit the gs_role_replication permission of the built-in role.

  Return type: void

- capture_view_to_json(text,integer)

  Description: Saves the view result to the directory specified by GUC: **perf_directory**. If **is_crossdb** is set to **1**, the view is accessed once for all databases. If the value of **is_crossdb** is **0**, the current database is accessed only once. Only users with the sysadmin or monitor admin permission can execute this function.

  Return type: int

- reset_unique_sql

  Description: Clears the unique SQL statements in the memory of the database node. (The sysadmin permission is required.)

  Return type: Boolean

Table 11-51 reset_unique_sql parameter description

Parameter	Type	Description
scope	text	Clearance scope type. The options are as follows: ● **GLOBAL**: Clears all nodes. If the value is **GLOBAL**, this function can be executed only on the primary node. ● **LOCAL**: Clears the current node.
clean_type	text	● **BY_USERID**: Unique SQL statements are cleared based on user IDs. ● **BY_CNID**: Unique SQL statements are cleared based on primary node IDs. ● **ALL**: All data is cleared.
clean_value	int8	Clearance value corresponding to the clearance type.

- wdr_xdb_query(db_name_str text, query text)

  Description: Provides the capability of executing local cross-database queries. For example, when connecting to the Postgres database, access tables in the **test** database.

  ```
  select col1 from wdr_xdb_query('dbname=test','select col1 from t1') as dd(col1 int);
  ```

  Return type: record

- pg_wlm_jump_queue(pid int)

  Description: Moves a task to the top of the queue of the primary node of the database.

  Return type: Boolean

  – **true**: success

  – **false**: failure

- gs_wlm_switch_cgroup(pid int, cgroup text)

  Description: Moves a job to another Cgroup to change the job priority.

  Return type: Boolean

  – **true**: success

  – **false**: failure

- pv_session_memctx_detail(threadid tid, MemoryContextName text)

  Description: Records information about the memory context **MemoryContextName** of the thread **tid** into the *threadid*_**timestamp.log** file in the *$GAUSSLOG*/**pg_log**/*${node_name}*/**dumpmem** directory. *threadid* can be obtained from *sessid* in the **GS_SESSION_MEMORY_DETAIL** view. In the officially released version, only the **MemoryContextName** that is an empty string (two single quotation marks indicate that the input is an empty string) is accepted. In this case, all memory context information is recorded. Otherwise, no operation is performed. For the DEBUG version for internal development and test personnel to debug, you can specify the **MemoryContextName** to be counted. In this case, all the memory usage of the context is recorded in the specified file. Only the administrator can execute this function.

  Return type: Boolean

  – **true**: success
  – **false**: failure

- pg_shared_memctx_detail(MemoryContextName text)

  Description: Records information about the memory context **MemoryContextName** into the *threadid*_**timestamp.log** file in the *$GAUSSLOG*/**pg_log**/*${node_name}*/**dumpmem** directory. This function is provided only for internal development and test personnel to debug in the DEBUG version. Calling this function in the officially released version does not involve any operation. Only the administrator can execute this function.

  Return type: Boolean

  – **true**: success
  – **false**: failure

- local_bgwriter_stat()

  Description: Displays the information about pages flushed by the bgwriter thread of this instance, number of pages in the candidate buffer chain, and buffer elimination information.

  Return type: record

- local_candidate_stat()

  Description: Displays the number of pages in the candidate buffer chain of this instance and buffer elimination information, including the normal buffer pool and segment buffer pool.

  Return type: record

- local_ckpt_stat()

  Description: Displays the information about checkpoints and flushing pages of the current instance.

  Return type: record

- local_double_write_stat()

  Description: Displays the doublewrite file status of the current instance.

  Return type: record

Table 11-52 local_double_write_stat parameters

Parameter	Type	Description
node_name	text	Instance name
curr_dwn	int8	Sequence number of the doublewrite file
curr_start_page	int8	Start page for restoring the doublewrite file
file_trunc_num	int8	Number of times that the doublewrite file is reused
file_reset_num	int8	Number of reset times after the doublewrite file is full
total_writes	int8	Total number of I/Os of the doublewrite file
low_threshold_writes	int8	Number of I/Os for writing doublewrite files with low efficiency (the number of I/O flushing pages at a time is less than 16)
high_threshold_writes	int8	Number of I/Os for writing doublewrite files with high efficiency (the number of I/O flushing pages at a time is more than 421)
total_pages	int8	Total number of pages that are flushed to the doublewrite file area
low_threshold_pages	int8	Number of pages that are flushed with low efficiency
high_threshold_pages	int8	Number of pages that are flushed with high efficiency
file_id	int8	ID of the current doublewrite file

- local_single_flush_dw_stat()

  Description: Displays the elimination of dual-write files on a single page in the instance.

  Return type: record
- local_pagewriter_stat()

  Description: Displays the page flushing information and checkpoint information of the current instance.

  Return type: record
- local_redo_stat()

  Description: Displays the replay status of the current standby instance.

  Return type: record

  Note: The returned replay status includes the current replay position and the replay position of the minimum restoration point.
- local_recovery_status()

  Description: Displays log flow control information about the primary and standby nodes.

Return type: record

- gs_wlm_node_recover(boolean isForce)

  Description: Obtains top SQL query statement-level statistics recorded in the current memory. If the input parameter is not **0**, the information is cleared from the memory.

  Return type: record

- gs_wlm_node_clean(cstring nodename)

  Description: Clears data after the dynamic load management node is faulty. (The current feature is a lab feature. Contact Huawei technical support before using it.) Only administrators can execute this function. This function is called by the database instance management module. You are not advised to directly call this function. This view is not supported in a centralized or standalone system.

  Return type: Boolean

- gs_cgroup_map_ng_conf(group name)

  Description: Reads the Cgroup configuration file of a specified logical database.

  Return type: record

- gs_wlm_switch_cgroup(sess_id int8, cgroup name)

  Description: Switches the Cgroup of a specified session.

  Return type: record

- comm_client_info()

  Description: Queries information about active client connections of a single node.

  Return type: SETOF record

- pg_sync_cstore_delta(text)

  Description: Synchronizes the delta table structure of a specified column-store table with that of the column-store primary table.

  Return type: bigint

- pg_sync_cstore_delta()

  Description: Synchronizes the delta table structure of all column-store tables with that of the column-store primary table.

  Return type: bigint

- pg_get_flush_lsn()

  Description: Returns the position of the Xlog flushed from the current node.

  Return type: text

- pg_get_sync_flush_lsn()

  Description: Returns the position of the Xlog flushed by the majority on the current node.

  Return type: text

- gs_create_log_tables()

  Description: Creates foreign tables and views for run logs and performance logs. (The current feature is a lab feature. Contact Huawei technical support before using it.)

Return type: void

Example:

```
openGauss=# select gs_create_log_tables();
 gs_create_log_tables
----------------------

(1 row)
```

- dbe_perf.get_global_full_sql_by_timestamp(start_timestamp timestamp with time zone, end_timestamp timestamp with time zone)

  Description: Obtains full SQL information at the database level. The result can be queried only in the system database but cannot be queried in the user database.

  Return type: record

  Table 11-53 dbe_perf.get_global_full_sql_by_timestamp parameter description

Parameter	Type	Description
start_timestamp	timestamp with time zone	Start point of the SQL start time range.
end_timestamp	timestamp with time zone	End point of the SQL start time range.

- dbe_perf.get_global_slow_sql_by_timestamp(start_timestamp timestamp with time zone, end_timestamp timestamp with time zone)

  Description: Obtains slow SQL information at the database level. The result can be queried only in the system database but cannot be queried in the user database.

  Return type: record

  Table 11-54 dbe_perf.get_global_slow_sql_by_timestamp parameter description

Parameter	Type	Description
start_timestamp	timestamp with time zone	Start point of the SQL start time range.
end_timestamp	timestamp with time zone	End point of the SQL start time range.

- statement_detail_decode(detail text, format text, pretty boolean)

  Description: Parses the **details** column in a full or slow SQL statement. The result can be queried only in the system database but cannot be queried in the user database.

  Return type: text

Table 11-55 statement_detail_decode parameter description

Parameter	Type	Description
detail	text	Set of events generated by the SQL statement (unreadable).
format	text	Parsing output format. The value is **plaintext**.
pretty	boolean	Whether to display the text in pretty format when **format** is set to **plaintext**. The options are as follows: ● The value **true** indicates that events are separated by **\n**. ● The value **false** indicates that events are separated by commas (,).

- pg_control_system()

  Description: Returns the status of the system control file.

  Return type: SETOF record
- pg_control_checkpoint()

  Description: Returns the system checkpoint status.

  Return type: SETOF record
- get_prepared_pending_xid

  Description: Returns nextxid when restoration is complete.

  Parameter: nan

  Return type: text
- pg_clean_region_info

  Description: Clears the regionmap.

  Parameter: nan

  Return type: character varying
- pg_get_delta_info

  Description: Obtains delta information from a single DN.

  Parameter: rel text, schema_name text

  Return type: part_name text, live_tuple bigint, data_size bigint, and blocknum bigint
- pg_get_replication_slot_name

  Description: Obtains the slot name.

  Parameter: nan

  Return type: text
- pg_get_running_xacts

  Description: Obtains running xact.

  Parameter: **nan**

  Return type: handle integer, gxid xid, state tinyint, node text, xmin xid, vacuum boolean, timeline bigint, prepare_xid xid, pid bigint, and next_xid xid

- pg_get_variable_info

  Description: Obtains the shared memory variable *cache*.

  Parameter: nan

  Return type: node_name text, nextOid oid, nextXid xid, oldestXid xid, xidVacLimit xid, oldestXidDB oid, lastExtendCSNLogpage xid, startExtendCSNLogpage xid, nextCommitSeqNo xid, latestCompletedXid xid, and startupMaxXid xid

- pg_get_xidlimit

  Description: Obtains transaction ID information from the shared memory.

  Parameter: nan

  Return type: nextXid xid, oldestXid xid, xidVacLimit xid, xidWarnLimit xid, xidStopLimit xid, xidWrapLimit xid, and oldestXidDB oid

- pg_relation_compression_ratio

  Description: Queries the compression rate of a table. By default, **1.0** is returned.

  Parameter: text

  Return type: real

- pg_relation_with_compression

  Description: Queries whether a table is compressed.

  Parameter: text

  Return type: Boolean

- pg_stat_file_recursive

  Description: Lists all files in the path.

  Parameter: location text

  Return type: path text, filename text, size bigint, and isdir boolean

- pg_stat_get_activity_for_temptable

  Description: Returns records of backend processes related to the temporary table.

  Parameter: nan

  Return type: datid oid, timelineid integer, tempid integer, and sessionid bigint

- pg_stat_get_activity_ng

  Description: Returns records of backend processes related to nodegroup.

  Parameter: pid bigint

  Return type: datid oid, pid bigint, sessionid bigint, and node_group text

- pg_stat_get_cgroup_info

  Description: Returns Cgroup information.

  Parameter: **nan**

  Return type: cgroup_name text, percent integer, usage_percent integer, shares bigint, usage bigint, cpuset text, relpath text, valid text, and node_group text

- pg_stat_get_realtime_info_internal

  Description: Returns real-time information. Currently, this API is unavailable. **FailedToGetSessionInfo** is returned.

  Parameter: oid, oid, bigint, cstring, oid

Return type: text

- pg_stat_get_workload_struct_info

  Description: Returns the load management data structure. (The current feature is a lab feature. Contact Huawei technical support before using it.)

  Parameter: nan

  Return type: text

- pg_test_err_contain_err

  Description: Tests the error type and return information.

  Parameter: integer

  Return type: void

- get_global_user_transaction()

  Description: Returns transaction information about each user on all nodes.

  Return type: node_name name, usename name, commit_counter bigint, rollback_counter bigint, resp_min bigint, resp_max bigint, resp_avg bigint, resp_total bigint, bg_commit_counter bigint, bg_rollback_counter bigint, bg_resp_min bigint, bg_resp_max bigint, bg_resp_avg bigint, and bg_resp_total bigint

- pg_collation_for

  Description: Returns the sorting rule corresponding to the input parameter string.

  Parameter: any (Explicit type conversion is required for constants.)

  Return type: text

- pgxc_unlock_for_sp_database(name Name)

  Description: Releases a specified database lock.

  Parameter: database name

  Return type: Boolean

- pgxc_lock_for_sp_database(name Name)

  Description: Locks a specified database.

  Parameter: database name

  Return type: Boolean

- copy_error_log_create()

  Description: Creates the error table (**public.pgxc_copy_error_log**) required for creating the **COPY FROM** error tolerance mechanism.

  Return type: Boolean

NOTE

- This function attempts to create the **public.pgxc_copy_error_log** table. For details about the table, see **Table 11-56**.
- In addition, it creates a B-tree index on the **relname** column and executes **REVOKE ALL on public.pgxc_copy_error_log FROM public** to manage permissions on the error table (the permissions are the same as those of the COPY statement).
- **public.pgxc_copy_error_log** is a row-store table. Therefore, this function can be executed and COPY error tolerance is available only when row-store tables can be created in the database instance. Note that after the GUC parameter **enable_hadoop_env** is enabled, row-store tables cannot be created in the database instance (the default value is **off** for GaussDB).
- Same as the error table and the COPY statement, the function requires sysadmin or higher permissions.
- If the **public.pgxc_copy_error_log** table or the **copy_error_log_relname_idx** index exists before the function creates it, the function will report an error and roll back.

Table 11-56 Error table public.pgxc_copy_error_log

Column	Type	Description
relname	character varying	Table name in the form of *Schema name*.*Table name*
begintime	timestamp with time zone	Time when a data format error was reported
filename	character varying	Name of the source data file where a data format error occurs
lineno	bigint	Number of the row where a data format error occurs in a source data file
rawrecord	text	Raw record of a data format error in the source data file
detail	text	Error details

- dynamic_func_control(scope text, function_name text, action text, "{params}" text[])

  Description: Dynamically enables built-in functions. Currently, only full SQL statements can be dynamically enabled.

  Return type: record

Table 11-57 Parameter description of dynamic_func_control

Parameter	Type	Description
scope	text	Scope where the function is to be dynamically enabled. Currently, only **LOCAL** is supported.
function_name	text	Function name. Currently, only **STMT** is supported.

Parameter	Type	Description
action	text	When **function_name** is set to **STMT**, the value of **action** can only be **TRACK**, **UNTRACK**, **LIST**, or **CLEAN**. ● **TRACK**: records the full SQL information of normalized SQL statements. ● **UNTRACK**: cancels the recording of full SQL information of normalized SQL statements. ● **LIST**: lists normalized SQL information that is recorded in the current track. ● **CLEAN**: cleans normalized SQL information that is recorded in the current track.
params	text[]	When **function_name** is set to **STMT**, the parameters corresponding to different actions are set as follows: ● **TRACK**: **'{"Normalized SQLID", "L0/L1/L2"}'** ● **UNTRACK**: **'{"Normalized SQLID"}'** ● LIST - '{}' ● CLEAN - '{}'

- gs_parse_page_bypath(path text, blocknum bigint, relation_type text, read_memory boolean)

  Description: Parses a specified table page and returns the path for storing the parsed content.

  Return type: text

  Note: Only the system administrator or O&M administrator can execute this function.

Table 11-58 gs_parse_page_bypath parameters

Parameter	Type	Description
path	text	● For an ordinary table or segment-page table, the relative path is *Tablespace name*/*Database OID*/*Relfilenode of the table (physical file name)*. For example, **base/16603/16394**. ● You can run the **pg_relation_filepath(table_name text)** command to query the relative path of the table file. To obtain the path of the partitioned table, view the **pg_partition** system catalog and call **pg_partition_filepath(partition_oid)**. ● Valid path formats are as follows: – global/relNode – base/dbNode/relNode – pg_tblspc/spcNode/version_dir/dbNode/relNode
blocknum	bigint	● **–1**: Information about all blocks (forcibly parsed from disks) ● **0**–*MaxBlockNumber*: Information about the corresponding block
relation_type	text	● **heap**: Astore table ● **uheap**: Ustore table ● **btree**: B-tree index ● **ubtree**: UB-tree index ● **segment**: Segment-page table ● **indexurq**: Ustore recycling queue
read_memory	boolean	● **false**: The system parses the page from the disk file. ● **true**: The system attempts to parse the page from the shared buffer. If the page does not exist in the shared buffer, the system parses the page from the disk file.

- gs_xlogdump_lsn(start_lsn text, end_lsn text)

  Description: Parses Xlogs within the specified LSN range and returns the path for storing the parsed content. You can use **pg_current_xlog_location()** to obtain the current Xlog position.

  Return type: text

  Parameters: LSN start position and LSN end position

  Note: Only the system administrator or O&M administrator can execute this function.

- gs_xlogdump_xid(c_xid xid)

  Description: Parses Xlogs of a specified XID and returns the path for storing the parsed content. You can use **txid_current()** to obtain the current transaction ID.

  Parameter: transaction ID

  Return type: text

  Note: Only the system administrator or O&M administrator can execute this function.

- gs_xlogdump_tablepath(path text, blocknum bigint, relation_type text)

  Description: Parses logs corresponding to a specified table page and returns the path for storing the parsed content.

  Return type: text

  Note: Only the system administrator or O&M administrator can execute this function.

Table 11-59 gs_xlogdump_tablepath parameters

Parameter	Type	Description
path	text	• For an ordinary table or segment-page table, the relative path is *Tablespace name/Database OID/Relfilenode of the table (physical file name)*. For example, **base/16603/16394**. • You can run the **pg_relation_filepath(table_name text)** command to query the relative path of the table file. To obtain the path of the partitioned table, view the **pg_partition** system catalog and call **pg_partition_filepath(partition_oid)**. • Valid path formats are as follows: – global/relNode – base/dbNode/relNode – pg_tblspc/spcNode/version_dir/dbNode/relNode
blocknum	bigint	• **–1**: Information about all blocks (forcibly parsed from disks) • **0**–*MaxBlockNumber*: Information about the corresponding block

Parameter	Type	Description
relation_type	text	● **heap**: Astore table ● **uheap**: Ustore table ● **btree**: B-tree index ● **ubtree**: UB-tree index ● **segment**: Segment-page table ● **indexurq**: Ustore recycling queue

- gs_xlogdump_parsepage_tablepath(path text, blocknum bigint, relation_type text, read_memory boolean)

  Description: Parses the specified table page and logs corresponding to the table page and returns the path for storing the parsed content. It can be regarded as one execution of **gs_parse_page_bypath** and **gs_xlogdump_tablepath**. The prerequisite for executing this function is that the table file exists. To view logs of deleted tables, call **gs_xlogdump_tablepath**.

  Return type: text

  Note: Only the system administrator or O&M administrator can execute this function.

Table 11-60 gs_xlogdump_parsepage_tablepath parameters

Parameter	Type	Description
path	text	● For an ordinary table or segment-page table, the relative path is *Tablespace name*/*Database OID*/*Relfilenode of the table (physical file name)*. For example, **base/16603/16394**. ● You can run the **pg_relation_filepath(table_name text)** command to query the relative path of the table file. To obtain the path of the partitioned table, view the **pg_partition** system catalog and call **pg_partition_filepath(partition_oid)**. ● Valid path formats are as follows: – global/relNode – base/dbNode/relNode – pg_tblspc/spcNode/version_dir/dbNode/relNode
blocknum	bigint	● **–1**: Information about all blocks (forcibly parsed from disks) ● **0**–*MaxBlockNumber*: Information about the corresponding block

Parameter	Type	Description
relation_type	text	● **heap**: Astore table ● **uheap**: Ustore table ● **btree**: B-tree index ● **ubtree**: UB-tree index ● **segment**: Segment-page table ● **indexurq**: Ustore recycling queue
read_memor y	boolean	● **false**: The system parses the page from the disk file. ● **true**: The system attempts to parse the page from the shared buffer. If the page does not exist in the shared buffer, the system parses the page from the disk file.

- gs_index_verify(Oid oid, uint32 blkno)

  Description: Checks whether the sequence of keys on the UB-tree index page or index tree is correct.

  Return type: record

Table 11-61 gs_index_verify parameters

Parameter	Type	Description
oid	Oid	● Index file relfilenode, which can be queried using **select relfilenode from pg_class where relname='***Index file name***'**.
blkno	uint32	● **0**: indicates that all pages in the index tree are checked. ● If the value is greater than 0, the index page whose page code is equal to the value of **blkno** is checked.

- gs_index_recycle_queue(Oid oid, int type, uint32 blkno)

  Description: Parses the UB-tree index recycling queue information.

  Return type: record

Table 11-62 gs_index_recycle_queue parameters

Parameter	Type	Description
oid	Oid	● Index file relfilenode, which can be queried using **select relfilenode from pg_class where relname='***Index file name***'**.

Parameter	Type	Description
type	int	● **0**: indicates that the entire queue to be recycled is parsed. ● **1**: indicates that the entire empty page queue is parsed. ● **2**: indicates that a single page is parsed.
blkno	uint32	● ID of the recycling queue page. This parameter is valid only when **type** is set to **2**. The value of **blkno** ranges from 1 to 4294967294.

- gs_stat_wal_entrytable(int64 idx)

  Description: Exports the content of the write-ahead log insertion status table in the Xlog.

  Return type: record

Table 11-63 gs_stat_wal_entrytable parameters

Category	Parameter	Type	Description
Input parameter	idx	int64	● **–1**: queries all elements in an array. ● **0-***Maximum value*: content of a specific array element.
Output parameter	idx	uint64	Records the subscripts in the corresponding array.
Output parameter	endlsn	uint64	Records the LSN label.
Output parameter	lrc	int32	Records the corresponding LRC.
Output parameter	status	uint32	Determines whether the Xlog corresponding to the current entry has been completely copied to the WAL buffer: ● **0**: Not copied. ● **1**: Copied

- gs_walwriter_flush_position()

  Description: Outputs the refresh position of write-ahead logs.

  Return type: record

Table 11-64 gs_walwriter_flush_position parameters

Category	Parameter	Type	Description
Output parameter	last_flush_status_entry	int32	Subscript index obtained after the Xlog flushes the tblEntry of the last flushed disk.
Output parameter	last_scanned_lrc	int32	LRC obtained after the Xlog flushes the last tblEntry scanned last time.
Output parameter	curr_lrc	int32	Latest LRC usage in the WALInsertStatusEntry status table. The LRC indicates the LRC value corresponding to the WALInsertStatusEntry when the next Xlog record is written.
Output parameter	curr_byte_pos	uint64	The latest Xlog position after the Xlog is written to the WAL file, which is also the next Xlog insertion point.
Output parameter	prev_byte_size	uint32	Length of the previous Xlog record.
Output parameter	flush_result	uint64	Position of the current global Xlog flush.
Output parameter	send_result	uint64	Xlog sending position on the current host.
Output parameter	shm_rqst_write_pos	uint64	The write position of the LogwrtRqst request in the XLogCtl recorded in the shared memory.
Output parameter	shm_rqst_flush_pos	uint64	The flush position of the LogwrtRqst request in the XLogCtl recorded in the shared memory.

Category	Parameter	Type	Description
Output parameter	shm_result_write_pos	uint64	The write position of the LogwrtResult request in the XLogCtl recorded in the shared memory.
Output parameter	shm_result_flush_pos	uint64	The flush position of the LogwrtResult request in the XLogCtl recorded in the shared memory.
Output parameter	curr_time	text	Current time.

- gs_walwriter_flush_stat(int operation)

  Description: Collects statistics on the frequency of writing and synchronizing write-ahead logs, data volume, and Xlog file information.

  Return type: record

Table 11-65 gs_walwriter_flush_stat parameters

Category	Parameter	Type	Description
Input parameter	operation	int	● **–1**: Disables the statistics function. (Default value) ● **0**: Enable the statistics function. ● **1**: Query statistics. ● **2**: Reset statistics.
Output parameter	write_times	uint64	Number of times that the Xlog calls the **write** API.
Output parameter	sync_times	uint64	Number of times that the Xlog calls the **sync** API.
Output parameter	total_xlog_sync_bytes	uint64	Total number of backend thread requests for writing data to Xlogs.

Category	Parameter	Type	Description
Output parameter	total_actual_xlog_sync_bytes	uint64	Total number of Xlogs that call the **sync** API for disk flushing.
Output parameter	avg_write_bytes	uint32	Number of Xlogs written each time the **XLogWrite** API is called.
Output parameter	avg_actual_write_bytes	uint32	Number of Xlogs written each time the **write** API is called.
Output parameter	avg_sync_bytes	uint32	Average number of Xlogs for each synchronization request.
Output parameter	avg_actual_sync_bytes	uint32	Actual Xlog amount of disk flushing by calling **sync** each time.
Output parameter	total_write_time	uint64	Total time of calling the write operation (unit: μs).
Output parameter	total_sync_time	uint64	Total time for calling the **sync** API (unit: μs).
Output parameter	avg_write_time	uint32	Average time for calling the **write** API each time (unit: μs).
Output parameter	avg_sync_time	uint32	Average time for calling the **sync** API each time (unit: μs).
Output parameter	curr_init_xlog_segno	uint64	ID of the latest Xlog segment file.

Category	Parameter	Type	Description
Output parameter	curr_open_xlog_segno	uint64	ID of the Xlog segment file that is being written.
Output parameter	last_reset_time	text	Time when statistics were last collected.
Output parameter	curr_time	text	Current time.

- gs_catalog_attribute_records()

  Description: Returns the definition of each field in a specified system catalog. Only common system catalogs whose OIDs are less than 10000 are supported. Indexes and TOAST tables are not supported.

  Parameter: OID of the system catalog

  Return type: record

- gs_comm_proxy_thread_status()

  Description: Collects statistics on data packets sent and received by the proxy communication library **comm_proxy** when a user-mode network is configured for the database instance.

  Parameter: nan

  Return type: record

  NOTE

  The query result of this function is displayed only when the user-mode network is deployed in a centralized environment and **enable_dfx in comm_proxy_attr** is set to **true**. In other scenarios, an error message is displayed, indicating that queries are not supported.

- pg_ls_tmpdir()

  Description: Returns the name, size, and last modification time of each file in the temporary directory (**pgsql_tmp**) of the default tablespace.

  Parameter: nan

  Return type: record

  Note: Only the system administrator or monitor administrator can execute this function.

Category	Parameter	Type	Description
Output parameter	name	text	File name

Output parameter	size	int8	File size (unit: byte)
Output parameter	modification	timestamptz	Last file modification time

- pg_ls_tmpdir(oid)

  Description: Returns the name, size, and last modification time of each file in the temporary directory (**pgsql_tmp**) of the specified tablespace.

  Parameter: oid

  Return type: record

  Note: Only the system administrator or monitor administrator can execute this function.

Category	Parameter	Type	Description
Input parameter	oid	oid	Tablespace ID
Output parameter	name	text	File name
Output parameter	size	int8	File size (unit: byte)
Output parameter	modification	timestamptz	Last file modification time

- pg_ls_waldir()

  Description: Returns the name, size, and last modification time of each file in the WAL directory.

  Parameter: nan

  Return type: record

  Note: Only the system administrator or monitor administrator can execute this function.

Category	Parameter	Type	Description
Output parameter	name	text	File name
Output parameter	size	int8	File size (unit: byte)
Output parameter	modification	timestamptz	Last file modification time

- gs_write_term_log(void)

Description: Writes a log to record the current **term** value of a DN. The standby DN returns **false**. After the data is successfully written to the primary DN, **true** is returned.

Return type: Boolean

11.5.26.13 Undo System Functions

- gs_undo_meta(type, zoneId, location)

  Description: Specifies metadata of each module in the undo system.

  Parameter description:

  – **type** (metadata type)

    The value **0** indicates the metadata corresponding to **Undo Zone(Record)**.

    The value **1** indicates the metadata corresponding to **Undo Zone(Transaction Slot)**.

    The value **2** indicates the metadata corresponding to **Undo Space(Record)**.

    The value **3** indicates the metadata corresponding to **Undo Space(Transaction Slot)**.

  – **zoneId** (undo zone ID)

    The value **–1** indicates the metadata of all undo zones.

    The value range 0–1024 x 1024 indicates the metadata of the corresponding zone ID.

  – **location** (read location)

    The value **0** indicates that data is read from the current memory.

    The value **1** indicates that data is read from a physical file.

  Return type: record

Table 11-66 Output example of gs_undo_meta(0,-1,0)

Category	Parameter	Type	Description
Output parameter	zoneId	oid	ID of the undo zone
Output parameter	persistType	oid	Persistence level
Output parameter	insert	text	Position of the next undo record to be inserted
Output parameter	discard	text	Position of the undo record that is recycled in common mode

Category	Parameter	Type	Description
Output parameter	end	text	Position of the undo record that is forcibly recycled. Values smaller than the value of this parameter indicate that the record has been recycled.
Output parameter	used	text	Used undo space
Output parameter	lsn	text	LSN to modify the zone
Output parameter	pid	oid	ID of the process bound to the zone

- gs_undo_translot(location, zoneId)

  Description: Specifies transaction slot information of the undo system.

  Parameter description:

  – **location** (read location)

    The value **0** indicates that data is read from the current memory.

    The value **1** indicates that data is read from a physical file.

  – **zoneId** (undo zone ID)

    The value **–1** indicates the metadata of all undo zones.

    The value range 0–1024 x 1024 indicates the metadata of the corresponding zone ID.

  Return type: record

Table 11-67 Output example of gs_undo_translot(0,-1)

Category	Parameter	Type	Description
Output parameter	groupId	oid	ID of the used undo zone
Output parameter	xactId	text	Transaction ID
Output parameter	startUndoPtr	text	Position where the undo record is inserted at the start of the transaction corresponding to a slot

Category	Parameter	Type	Description
Output parameter	endUndoPtr	text	Position where the undo record is inserted at the end of the transaction corresponding to a slot
Output parameter	lsn	text	Pointer to the corresponding slot
Output parameter	slot_states	oid	Transaction state. The options are as follows: **0**: committed; **1**: being executed; **2**: being rolled back; **3**: rolled back

- gs_stat_undo()

  Description: Undo statistics.

  Return type: record

Table 11-68 gs_stat_undo parameters

Category	Parameter	Type	Description
Output parameter	curr_used_zone_count	uint32	Number of used undo zones.
Output parameter	top_used_zones	text	Information about the first three undo zones with the maximum usage. The output format is as follows: **(zoneId1:***Used size***; zoneId2:***Used size***; zoneId3:***Used size***)**.
Output parameter	curr_used_undo_size	uint32	Total size of the undo tablespace that is being used. The unit is MB.
Output parameter	undo_threshold	uint32	Calculation result of the value of the GUC parameter **undo_space_limit_size** x 80%. The unit is MB.
Output parameter	oldest_xid_in_undo	uint64	XID of the transaction recycled to the undo space. The undo records generated by the transaction whose XID is smaller than the value of XID are recycled.

Category	Parameter	Type	Description
Output parameter	oldest_xmin	uint64	Oldest active transaction.
Output parameter	total_undo_chain_len	int64	Total length of all accessed undo chains.
Output parameter	max_undo_chain_len	int64	Maximum length of the accessed undo chain.
Output parameter	create_undo_file_count	uint32	Number of created undo files.
Output parameter	discard_undo_file_count	uint32	Number of deleted undo files.

- gs_undo_record(undoptr)

  Description: Undo record resolution.

  Parameter description:

  – **undoptr** (undo record pointer)

  Return type: record

- gs_undo_dump_parsepage_mv(relpath text, blkno bigint, reltype text, rmem boolean)

  Description: Parses the page header information of the disk page in the Ustore table, header information of each tuple, flag bit information, and all historical undo version information that can be queried.

  Return type: text

  Note: Only the system administrator or O&M administrator can execute this function.

  NOTE

  Currently, this interface supports only Ustore tables.

Table 11-69 gs_undo_dump_parsepage_mv parameters

Category	Parameter	Type	Description
Input parameter	relpath	text	Relative path of the Ustore table data file, in the format of *Tablespace name*/*Database OID*/*relfilenode*. For example, **base/16603/16384**. You can run the **pg_relation_filepath('tablename')** command to query the relative path.
Input parameter	blkno	bigint	● **–1**: All block pages are parsed. ● **0**–*MaxBlocNumber*: A specified block page is parsed.
Input parameter	reltype	text	Table type. Currently, only the Ustore table is supported. The value is **uheap**.
Input parameter	rmem	boolean	● false ● true Currently, the value can only be **false**, indicating that the corresponding page is parsed from the disk file.
Output parameter	output	text	Absolute path of the parsing result file.

- gs_undo_meta_dump_zone(zone_id int, read_memory boolean)

  Description: Parses zone metadata in the undo module.

  Return type: record

Table 11-70 gs_undo_meta_dump_zone parameters

Category	Parameter	Type	Description
Input parameter	zone_id	int	Undo zone ID ● **–1**: All undo zones are queried. ● **0–1048575**: The undo zone metadata is queried based on the zone ID.
Input parameter	read_memory	boolean	● **true**: Data is read from the current memory. ● **false**: Data is read from the physical file.

Category	Parameter	Type	Description
Output parameter	zone_id	int	Undo zone ID
Output parameter	persist_type	int	Persistence level ● **0**: ordinary table ● **1**: unlogged table ● **2**: temporary table
Output parameter	insert	text	Position of the next undo record to be inserted
Output parameter	discard	text	Position of the undo record that is recycled in common mode
Output parameter	forcediscard	text	Position of the undo record that is forcibly recycled. Values smaller than the value of this parameter indicate that the record has been recycled.
Output parameter	lsn	text	LSN to modify the zone

- gs_undo_meta_dump_spaces(zone_id int, read_memory boolean)

  Description: Parses the metadata of the undo record space and undo slot space in the undo module.

  Return type: record

Table 11-71 gs_undo_meta_dump_spaces parameters

Category	Parameter	Type	Description
Input parameter	zone_id	int	Undo zone ID ● **–1**: All undo zones are queried. ● **0–1048575**: The undo zone metadata is queried based on the zone ID.

Category	Parameter	Type	Description
Input parameter	read_memory	boolean	• **true**: Data is read from the current memory. • **false**: Data is read from the physical file.
Output parameter	zone_id	int	Undo zone ID
Output parameter	undorecord_space_tail	text	End position of the undo record space
Output parameter	undorecord_space_head	text	Start position of the undo record space
Output parameter	undorecord_space_lsn	text	LSN of the modified undo record space
Output parameter	undoslot_space_tail	text	End position of the undo slot space
Output parameter	undoslot_space_head	text	Start position of the undo slot space
Output parameter	undoreslot_space_lsn	text	LSN of the modified undo slot space

- gs_undo_meta_dump_slot(zone_id int, read_memory boolean)
  Description: Parses slot metadata in the undo module.
  Return type: record

Table 11-72 gs_undo_meta_dump_slot parameters

Category	Parameter	Type	Description
Input parameter	zone_id	int	Undo zone ID ● **–1**: All undo zones are queried. ● **0–1048575**: The undo zone metadata is queried based on the zone ID.
Input parameter	read_memory	boolean	● **true**: Data is read from the current memory. ● **false**: Data is read from the physical file.
Output parameter	zone_id	int	Undo zone ID
Output parameter	allocate	text	Allocation position of the undo transaction slot
Output parameter	recycle	text	Recycling position of the undo transaction slot
Output parameter	frozen_xid	text	Frozen XID, which is used to determine the visibility
Output parameter	global_frozen_xid	text	Minimum frozen XID in the system. Transactions whose XID is smaller than the value of this parameter are visible.
Output parameter	recycle_xid	text	Recycled XID. Transactions whose XID is smaller than the value of this parameter are recycled.
Output parameter	global_recycle_xid	text	Minimum recycled XID in the system. Transactions whose XID is smaller than the value of this parameter are recycled.

- gs_undo_translot_dump_slot(zone_id int, read_memory boolean)
  Description: Parses slots in a zone.
  Return type: record

Table 11-73 gs_undo_translot_dump_slot parameters

Category	Parameter	Type	Description
Input parameter	zone_id	oid	Undo zone ID ● **–1**: All undo zones are queried. ● **0–1048575**: The undo zone metadata is queried based on the zone ID.
Input parameter	read_memory	boolean	● **true**: Data is read from the current memory. ● **false**: Data is read from the physical file.
Output parameter	zone_id	text	Undo zone ID
Output parameter	slot_xid	text	Transaction ID
Output parameter	start_undoptr	text	Position where the undo record is inserted at the start of the transaction corresponding to a slot
Output parameter	end_undoptr	text	Position where the undo record is inserted at the end of the transaction corresponding to a slot
Output parameter	lsn	text	LSN of the modified slot
Output parameter	slot_states	oid	Transaction state ● **0**: committed ● **1**: being executed ● **2**: being rolled back ● **3**: rolled back

- gs_undo_translot_dump_xid(slot_xid xid, read_memory boolean)

  Description: Parses the slot in a zone based on the XID.

  Return type: record

Table 11-74 gs_undo_translot_dump_xid parameters

Category	Parameter	Type	Description
Input parameter	slot_xid	xid	XID to be queried
Input parameter	read_memory	boolean	● **true**: Data is read from the current memory. ● **false**: Data is read from the physical file.
Output parameter	zone_id	text	Undo zone ID
Output parameter	slot_xid	text	Transaction ID
Output parameter	start_undoptr	text	Position where the undo record is inserted at the start of the transaction corresponding to a slot
Output parameter	end_undoptr	text	Position where the undo record is inserted at the end of the transaction corresponding to a slot
Output parameter	lsn	text	LSN of the modified slot
Output parameter	slot_states	oid	Transaction state ● **0**: committed ● **1**: being executed ● **2**: being rolled back ● **3**: rolled back

- gs_undo_dump_record(undoptr bigint)

  Description: Parses undo records.

  Return type: record

Table 11-75 gs_undo_dump_record parameters

Category	Parameter	Type	Description
Input parameter	undoptr	bigint	Start position of the undo record to be parsed

Category	Parameter	Type	Description
Output parameter	undoptr	bigint	Start position of the undo record to be parsed
Output parameter	xactid	text	Transaction ID
Output parameter	cid	text	command id
Output parameter	reloid	text	relation oid
Output parameter	relfilenode	text	Relfinode of the file
Output parameter	utype	text	Undo record type
Output parameter	blkprev	text	Position of the previous undo record in the same block
Output parameter	blockno	text	Block number
Output parameter	uoffset	text	Undo record offset
Output parameter	prevurp	text	Position of the previous undo record
Output parameter	payloadlen	text	Length of the undo record data
Output parameter	oldxactid	text	Previous XID
Output parameter	partitionoid	text	Partition OID

Category	Parameter	Type	Description
Output parameter	tablespace	text	Tablespace
Output parameter	alreadyread_bytes	text	Length of the read undo record
Output parameter	prev_undorec_len	text	Length of the previous undo record
Output parameter	td_id	text	ID of the transaction directory
Output parameter	reserved	text	Whether to reserve the record
Output parameter	flag	text	Flag 1
Output parameter	flag2	text	Flag 2
Output parameter	t_hoff	text	Length of the undo record data header

- gs_undo_dump_xid(undo_xid xid)

  Description: Parses undo records based on the XID.

  Return type: record

Table 11-76 gs_undo_dump_xid parameters

Category	Parameter	Type	Description
Input parameter	undo_xid	xid	XID
Output parameter	undoptr	xid	Start position of the undo record to be parsed

Category	Parameter	Type	Description
Output parameter	xactid	text	XID
Output parameter	cid	text	command id
Output parameter	reloid	text	relation oid
Output parameter	relfilenode	text	Relfinode of the file
Output parameter	utype	text	Undo record type
Output parameter	blkprev	text	Position of the previous undo record in the same block
Output parameter	blockno	text	Block number
Output parameter	uoffset	text	Undo record offset
Output parameter	prevurp	text	Position of the previous undo record
Output parameter	payloadlen	text	Length of the undo record data
Output parameter	oldxactid	text	Previous XID
Output parameter	partitionoid	text	Partition OID
Output parameter	tablespace	text	Tablespace

Category	Parameter	Type	Description
Output parameter	alreadyread_bytes	text	Length of the read undo record
Output parameter	prev_undorec_len	text	Length of the previous undo record
Output parameter	td_id	text	ID of the transaction directory
Output parameter	reserved	text	Whether to reserve the record
Output parameter	flag	text	Flag 1
Output parameter	flag2	text	Flag 2
Output parameter	t_hoff	text	Length of the undo record data header

- gs_verify_undo_record(type, startIdx, endIdx, location)

  Description: Verifies the Undo record.

  Return type: record

  NOTE

  This interface is reserved for expansion and cannot be used.

- gs_verify_undo_translot(type, startIdx, endIdx, location)

  Description: Verifies the Undo transaction slot.

  Return type: record

  NOTE

  This interface is reserved for expansion and cannot be used.

- gs_verify_undo_meta(type, startIdx, endIdx, location)

  Description: Verifies the Undo meta information.

  Return type: record

  NOTE

  This interface is reserved for expansion and cannot be used.

11.5.27 Statistics Information Functions

Statistics information functions are divided into the following two categories: functions that access databases, using the OID of each table or index in a database to mark the database for which statistics are generated; functions that access servers, identified by the server process ID, whose value ranges from 1 to the number of currently active servers.

- pg_stat_get_db_conflict_tablespace(oid)

  Description: Specifies the number of queries canceled due to a conflict between the restored tablespace and the deleted tablespace in the database.

  Return type: bigint
- pg_control_group_config

  Description: Prints Cgroup configurations on the current node.

  Return type: record
- pg_stat_get_db_stat_reset_time(oid)

  Description: Specifies the most recent time when database statistics were reset. It is initialized to the system time during the first connection to each database. The reset time is updated when you call **pg_stat_reset** on the database and execute **pg_stat_reset_single_table_counters** against any table or index in it.

  Return type: timestamptz
- pg_stat_get_function_total_time(oid)

  Description: Specifies the total wall clock time spent in the function, in microseconds. The time spent on this function calling other functions is included.

  Return type: bigint
- pg_stat_get_xact_tuples_returned(oid)

  Description: Specifies the number of rows read through sequential scans when the parameter is a table in the current transaction or the number of index entries returned when the parameter is an index.

  Return type: bigint
- pg_lock_status()

  Description: Queries information about locks held by open transactions. All users can execute this function.

  Return type: For details, see **PG_LOCKS** which is obtained by querying this function.
- pg_stat_get_xact_numscans(oid)

  Description: Specifies the number of sequential scans performed when the parameter is a table in the current transaction or the number of index scans performed when the parameter is an index.

  Return type: bigint
- pg_stat_get_xact_blocks_fetched(oid)

  Description: Specifies the number of disk block fetch requests for a table or an index in the current transaction.

  Return type: bigint

- pg_stat_get_xact_blocks_hit(oid)

  Description: Specifies the number of disk block fetch requests for tables or indexes found in cache in the current transaction.

  Return type: bigint
- pg_stat_get_xact_function_calls(oid)

  Description: Specifies the number of times the function is called in the current transaction.

  Return type: bigint
- pg_stat_get_xact_function_self_time(oid)

  Description: Specifies the time spent on this function in the current transaction, excluding the time spent on this function internally calling other functions.

  Return type: bigint
- pg_stat_get_xact_function_total_time(oid)

  Description: Specifies the total wall clock time (in microseconds) spent on the function in the current transaction, including the time spent on this function internally calling other functions.

  Return type: bigint
- pg_stat_get_wal_senders()

  Description: Queries walsender information on the primary node.

  Return type: setofrecord

  The following table describes return columns.

Table 11-77 Return column description

Column	Type	Description
pid	bigint	Thread ID of the WAL sender
sender_pid	integer	Lightweight thread ID of the WAL sender
local_role	text	Type of the primary node
peer_role	text	Type of the standby node
peer_state	text	Status of the standby node
state	text	Status of the WAL sender
catchup_start	timestamp with time zone	Startup time of a catchup task
catchup_end	timestamp with time zone	End time of a catchup task

Column	Type	Description
sender_sent_location	text	Sending position of the primary node
sender_write_location	text	Writing position of the primary node
sender_flush_location	text	Flushing position of the primary node
sender_replay_location	text	Redo position of the primary node
receiver_received_location	text	Receiving position of the standby node
receiver_write_location	text	Writing position of the standby node
receiver_flush_location	text	Flushing position of the standby node
receiver_replay_location	text	Redo position of the standby node
sync_percent	text	Synchronization percentage
sync_state	text	Synchronization status
sync_group	text	Group to which the synchronous replication belongs
sync_priority	text	Priority of synchronous replication
sync_most_available	text	Maximum availability mode
channel	text	Channel information of the WAL sender

- get_paxos_replication_info()

  Description: Queries the primary/standby replication status in Paxos mode.

  Return type: setofrecord

  The following table describes return columns.

Table 11-78 Return column description

Column	Type	Description
paxos_write_location	text	Location of the Xlog that has been written to the Distribute Consensus Framework (DCF)
paxos_commit_location	text	Location of the Xlog agreed in the DCF
local_write_location	text	Writing position of a node
local_flush_location	text	Flushing position of a node
local_replay_location	text	Redo position of a node
dcf_replication_info	text	DCF module information of a node

- pg_stat_get_stream_replications()
  Description: Queries the primary/standby replication status.
  Return type: setofrecord
  The following table describes return values.

Table 11-79 Return value description

Return Parameter	Type	Description
local_role	text	Local role
static_connections	integer	Connection statistics
db_state	text	Database status
detail_information	text	Detailed information

- pg_stat_get_db_numbackends(oid)
  Description: Specifies the number of active server processes for a database.
  Return type: integer
- pg_stat_get_db_xact_commit(oid)
  Description: Specifies the number of transactions committed in a database.
  Return type: bigint
- pg_stat_get_db_xact_rollback(oid)
  Description: Specifies the number of transactions rolled back in a database.
  Return type: bigint
- pg_stat_get_db_blocks_fetched(oid)
  Description: Specifies the number of disk blocks fetch requests for a database.

Return type: bigint

- pg_stat_get_db_blocks_hit(oid)

  Description: Specifies the number of disk block fetch requests found in cache for a database.

  Return type: bigint

- pg_stat_get_db_tuples_returned(oid)

  Description: Specifies the number of tuples returned for a database.

  Return type: bigint

- pg_stat_get_db_tuples_fetched(oid)

  Description: Specifies the number of tuples fetched for a database.

  Return type: bigint

- pg_stat_get_db_tuples_inserted(oid)

  Description: Specifies the number of tuples inserted in a database.

  Return type: bigint

- pg_stat_get_db_tuples_updated(oid)

  Description: Specifies the number of tuples updated in a database.

  Return type: bigint

- pg_stat_get_db_tuples_deleted(oid)

  Description: Specifies the number of tuples deleted in a database.

  Return type: bigint

- pg_stat_get_db_conflict_lock(oid)

  Description: Specifies the number of lock conflicts in a database.

  Return type: bigint

- pg_stat_get_db_deadlocks(oid)

  Description: Specifies the number of deadlocks in a database.

  Return type: bigint

- pg_stat_get_numscans(oid)

  Description: Specifies the number of sequential row scans done if parameters are in a table or the number of index scans done if parameters are in an index.

  Return type: bigint

- pg_stat_get_role_name(oid)

  Description: Obtains the username based on the user OID. Only users with the sysadmin or monitor admin permission can access the information.

  Return type: text

  Example:

```
openGauss=# select pg_stat_get_role_name(10);
 pg_stat_get_role_name
-----------------------
 aabbcc
(1 row)
```

- pg_stat_get_tuples_returned(oid)

Description: Specifies the number of sequential row scans done if parameters are in a table or the number of index scans done if parameters are in an index.

Return type: bigint

- pg_stat_get_tuples_fetched(oid)

  Description: Specifies the number of table rows fetched by bitmap scans if parameters are in a table or the number of table rows fetched by simple index scans using the index if parameters are in an index.

  Return type: bigint

- pg_stat_get_tuples_inserted(oid)

  Description: Specifies the number of rows inserted into a table.

  Return type: bigint

- pg_stat_get_tuples_updated(oid)

  Description: Specifies the number of rows updated in a table.

  Return type: bigint

- pg_stat_get_tuples_deleted(oid)

  Description: Specifies the number of rows deleted from a table.

  Return type: bigint

- pg_stat_get_tuples_changed(oid)

  Description: Specifies the total number of inserted, updated, and deleted rows after a table was last analyzed or autoanalyzed.

  Return type: bigint

- pg_stat_get_tuples_hot_updated(oid)

  Description: Specifies the number of rows hot updated in a table.

  Return type: bigint

- pg_stat_get_live_tuples(oid)

  Description: Specifies the number of live rows in a table.

  Return type: bigint

- pg_stat_get_dead_tuples(oid)

  Description: Specifies the number of dead rows in a table.

  Return type: bigint

- pg_stat_get_blocks_fetched(oid)

  Description: Specifies the number of disk block fetch requests for a table or an index.

  Return type: bigint

- pg_stat_get_blocks_hit(oid)

  Description: Specifies the number of disk block requests found in cache for a table or an index.

  Return type: bigint

- pg_stat_get_partition_tuples_inserted(oid)

  Description: Specifies the number of rows in the corresponding table partition.

  Return type: bigint

- pg_stat_get_partition_tuples_updated(oid)

  Description: Specifies the number of rows that have been updated in the corresponding table partition.

  Return type: bigint
- pg_stat_get_partition_tuples_deleted(oid)

  Description: Specifies the number of rows deleted from the corresponding table partition.

  Return type: bigint
- pg_stat_get_partition_tuples_changed(oid)

  Description: Specifies the total number of inserted, updated, and deleted rows after a table partition was last analyzed or autoanalyzed.

  Return type: bigint
- pg_stat_get_partition_live_tuples(oid)

  Description: Specifies the number of live rows in a partitioned table.

  Return type: bigint
- pg_stat_get_partition_dead_tuples(oid)

  Description: Specifies the number of dead rows in a partitioned table.

  Return type: bigint
- pg_stat_get_xact_tuples_fetched(oid)

  Description: Specifies the number of tuple rows scanned in a transaction.

  Return type: bigint
- pg_stat_get_xact_tuples_inserted(oid)

  Description: Specifies the number of tuple inserted into the active subtransactions related to a table.

  Return type: bigint
- pg_stat_get_xact_tuples_deleted(oid)

  Description: Specifies the number of deleted tuples in the active subtransactions related to a table.

  Return type: bigint
- pg_stat_get_xact_tuples_hot_updated(oid)

  Description: Specifies the number of hot updated tuples in the active subtransactions related to a table.

  Return type: bigint
- pg_stat_get_xact_tuples_updated(oid)

  Description: Specifies the number of updated tuples in the active subtransactions related to a table.

  Return type: bigint
- pg_stat_get_xact_partition_tuples_inserted(oid)

  Description: Specifies the number of inserted tuples in the active subtransactions related to a table partition.

  Return type: bigint
- pg_stat_get_xact_partition_tuples_deleted(oid)

Description: Specifies the number of deleted tuples in the active subtransactions related to a table partition.

Return type: bigint

- pg_stat_get_xact_partition_tuples_hot_updated(oid)

  Description: Specifies the number of hot updated tuples in the active subtransactions related to a table partition.

  Return type: bigint

- pg_stat_get_xact_partition_tuples_updated(oid)

  Description: Specifies the number of updated tuples in the active subtransactions related to a table partition.

  Return type: bigint

- pg_stat_get_last_vacuum_time(oid)

  Description: Specifies the most recent time when the autovacuum thread is manually started to clear a table.

  Return type: timestamptz

- pg_stat_get_last_autovacuum_time(oid)

  Description: Specifies the time of the last vacuum initiated by the autovacuum daemon on a table.

  Return type: timestamptz

- pg_stat_get_vacuum_count(oid)

  Description: Specifies the number of times a table is manually cleared.

  Return type: bigint

- pg_stat_get_autovacuum_count(oid)

  Description: Specifies the number of times the autovacuum daemon is started to clear a table.

  Return type: bigint

- pg_stat_get_last_analyze_time(oid)

  Description: Specifies the last time when a table starts to be analyzed manually or by the autovacuum thread.

  Return type: timestamptz

- pg_stat_get_last_autoanalyze_time(oid)

  Description: Specifies the time when the last analysis initiated by the autovacuum daemon on a table.

  Return type: timestamptz

- pg_stat_get_analyze_count(oid)

  Description: Specifies the number of times a table is manually analyzed.

  Return type: bigint

- pg_stat_get_autoanalyze_count(oid)

  Description: Specifies the number of times the autovacuum daemon analyzes a table.

  Return type: bigint

- pg_total_autovac_tuples(bool,bool)

Description: Returns tuple records related to the total autovac, such as **nodename**, **nspname**, **relname**, and tuple IUDs. The input parameters specify whether to query **relation** and **local** information, respectively.

Return type: setofrecord

The following table describes return parameters.

Table 11-80 Return parameter description

Return Parameter	Type	Description
nodename	name	Node name
nspname	name	Name of a namespace
relname	name	Name of an object, such as a table, an index, or a view
partname	name	Partition name
n_dead_tuples	bigint	Number of dead rows in a table partition
n_live_tuples	bigint	Number of live rows in a table partition
changes_since_analyze	bigint	Number of changes generated by ANALYZE

- pg_autovac_status(oid)

  Description: Returns autovac information, such as **nodename**, **nspname**, **relname**, **analyze**, **vacuum**, thresholds of **analyze** and **vacuum**, and the number of analyzed or vacuumed tuples. Only users with the sysadmin permission can use this function.

  Return type: setofrecord

  The following table describes return parameters.

Table 11-81 Return parameter description

Return Parameter	Type	Description
nspname	text	Name of a namespace
relname	text	Name of an object, such as a table, an index, or a view
nodename	text	Node name
doanalyze	Boolean	Whether to execute **ANALYZE**
anltuples	bigint	Number of ANALYZE tuples

Return Parameter	Type	Description
anlthresh	bigint	ANALYZE threshold
dovacuum	Boolean	Whether to execute **VACUUM**
vactuples	bigint	Number of VACUUM tuples
vacthresh	bigint	VACUUM threshold

- pg_autovac_timeout(oid)

  Description: Returns the number of consecutive timeouts during the autovac operation on a table. If the table information is invalid or the node information is abnormal, **NULL** will be returned.

  Return type: bigint
- pg_stat_get_last_data_changed_time(oid)

  Description: Returns the time when **INSERT**, **UPDATE**, **DELETE**, or **EXCHANGE**/**TRUNCATE**/**DROP PARTITION** was last performed on a table. The data in the **last_data_changed** column of the **PG_STAT_ALL_TABLES** view is calculated by using this function. The performance of obtaining the last modification time by using the view is poor when the table has a large amount of data. In this case, you are advised to use the function.

  Return type: timestamptz
- pg_stat_set_last_data_changed_time(oid)

  Description: Manually changes the time when **INSERT**, **UPDATE**, **DELETE**, or **EXCHANGE**/**TRUNCATE**/**DROP PARTITION** was last performed.

  Return type: void
- pg_backend_pid()

  Description: Specifies the thread ID of the server thread attached to the current session.

  Return type: integer
- pg_stat_get_activity(integer)

  Description: Returns a record about the backend with the specified PID. A record for each active backend in the system is returned if **NULL** is specified. The returned result does not contain the **connection_info** column. The initial user, system administrators and users with the monadmin permission can view all data. Common users can only query their own results.

  Example:

```
openGauss=# select * from pg_stat_get_activity(139881386280704);
 datid |       pid       | sessionid | usesysid | application_name | state  |
query                          | waiting |          xact_start           |          query_start          |
    backend_start          |         state_change          | client_addr | client_hostname | client_port | enqueue
|     query_id      |   srespool   | global_sessionid | unique_sql_id | trace_id
-------+-----------------+-----------+----------+------------------+--------
+------------------------------------------------------+---------+-------------------------------
+-------------------------------+-----
-------------------------+-----------------------------+-------------+-----------------+-------------+---------
+-------------------+--------------+------------------+---------------+----------
 16545 | 139881386280704 |        69 |       10 | gsql             | active | select * from
```

```
pg_stat_get_activity(139881386280704); | f       | 2022-01-18 19:43:05.167718+08 | 2022-01-18
19:43:05.167718+08 | 2022
-01-18 19:42:33.513507+08 | 2022-01-18 19:43:05.16773+08 |             |               |          -1 |         |
72620543991624410 | default_pool | 1938253334#69#0  |    3751941862 |
(1 row)
```

Return type: setofrecord

The following table describes return parameters.

Table 11-82 Return parameter description

Return Parameter	Type	Description
datid	oid	OID of the database that the user session connects to in the backend
pid	bigint	Backend thread ID
sessionid	bigint	Session ID
usesysid	oid	OID of the user logged in to the backend
application_name	text	Name of the application connected to the backend
state	text	Overall status of the backend
query	text	Latest query at the backend. If **state** is **active**, this column shows the ongoing query. In all other states, it shows the last query that was executed.
waiting	Boolean	Whether the backend is currently waiting on a lock. If yes, the value is **true**.
xact_start	timestamp with time zone	Time when current transaction was started (null if no transaction is active). If the current query is the first of its transaction, the value of this column is the same as that of the **query_start** column.

Return Parameter	Type	Description
query_start	timestamp with time zone	Time when the currently active query was started, or time when the last query was started if **state** is not **active**
backend_start	timestamp with time zone	Time when this process was started, that is, when the client connected to the server
state_change	timestamp with time zone	Time when **state** was last modified
client_addr	inet	IP address of the client connected to the backend. If this column is **NULL**, it indicates either the client is connected via a Unix socket on the server or this is an internal process, such as **AUTOVACUUM**.
client_hostname	text	Host name of the connected client, as reported by a reverse DNS lookup of **client_addr**. This column will be non-null only for IP connections and only when **log_hostname** is enabled.
client_port	integer	TCP port number that the client uses for communication with this backend (**–1** if a Unix socket is used)
enqueue	text	Unsupported currently
query_id	bigint	ID of a query
srespool	name	Name of the resource pool
global_sessionid	text	Global session ID

Return Parameter	Type	Description
unique_sql_id	bigint	Unique SQL statement ID
trace_id	text	Driver-specific trace ID, which is associated with an application request

- pg_stat_get_activity_with_conninfo(integer)

  Description: Returns a record about the backend with the specified PID. A record for each active backend in the system is returned if **NULL** is specified. The initial user, system administrators and users with the monadmin permission can view all data. Common users can only query their own results.

  Return type: setofrecord

  The following table describes return values.

Table 11-83 Return value description

Return Value	Return Type	Description
datid	oid	OID of the database that the user session connects to in the backend
pid	bigint	Backend thread ID
sessionid	bigint	Session ID
usesysid	oid	OID of the user logged in to the backend
application_name	text	Name of the application connected to the backend
state	text	Overall status of the backend
query	text	Latest query at the backend. If **state** is **active**, this column shows the ongoing query. In all other states, it shows the last query that was executed.

Return Value	Return Type	Description
waiting	Boolean	Whether the backend is currently waiting on a lock. If yes, the value is **true**.
xact_start	timestamp with time zone	Time when current transaction was started (null if no transaction is active). If the current query is the first of its transaction, the value of this column is the same as that of the **query_start** column.
query_start	timestamp with time zone	Time when the currently active query was started, or time when the last query was started if **state** is not **active**
backend_start	timestamp with time zone	Time when this process was started, that is, when the client connected to the server
state_change	timestamp with time zone	Time when **state** was last modified
client_addr	inet	IP address of the client connected to the backend If this column is **NULL**, it indicates either the client is connected via a Unix socket on the server or this is an internal process, such as **AUTOVACUUM**.
client_hostname	text	Host name of the connected client, as reported by a reverse DNS lookup of **client_addr**. This column will be non-null only for IP connections and only when **log_hostname** is enabled.

Return Value	Return Type	Description
client_port	integer	TCP port number that the client uses for communication with this backend (**–1** if a Unix socket is used)
enqueue	text	Unsupported currently
query_id	bigint	ID of a query
connection_info	text	A string in JSON format recording the driver type, driver version, driver deployment path, and process owner of the connected database
srespool	name	Name of the resource pool
global_sessionid	text	Global session ID
unique_sql_id	bigint	Unique SQL statement ID
trace_id	text	Driver-specific trace ID, which is associated with an application request

- pg_user_iostat(text)

  Description: Displays the I/O load management information about the job currently executed by the user. (The current feature is a lab feature. Contact Huawei technical support before using it.)

  Return type: record

  The following table describes return fields.

Name	Type	Description
userid	oid	User ID
min_curr_iops	int4	Minimum I/O of the current user across database nodes. The IOPS is counted by ones for column storage and by ten thousands for row storage.
max_curr_iops	int4	Maximum I/O of the current user across database nodes. The IOPS is counted by ones for column storage and by ten thousands for row storage.

Name	Type	Description
min_peak_iops	int4	Minimum peak I/O of the current user across database nodes. The IOPS is counted by ones for column storage and by ten thousands for row storage.
max_peak_iops	int4	Maximum peak I/O of the current user across database nodes. The IOPS is counted by ones for column storage and by ten thousands for row storage.
io_limits	int4	**io_limits** set for the resource pool specified by the user. The IOPS is counted by ones for column storage and by ten thousands for row storage.
io_priority	text	**io_priority** set for the user. The IOPS is counted by ones for column storage and by ten thousands for row storage.
curr_io_limits	int4	Real-time **io_limits** value when **io_priority** is used to control I/Os

- pg_stat_get_function_calls(oid)

  Description: Specifies the number of times the function has been called.

  Return type: bigint
- pg_stat_get_function_self_time(oid)

  Description: Specifies the time spent in only this function. The time spent on this function calling other functions is excluded.

  Return type: bigint
- pg_stat_get_backend_idset()

  Description: Sets the number of currently active server processes (from 1 to the number of active server processes).

  Return type: setofinteger
- pg_stat_get_backend_pid(integer)

  Description: Specifies the ID of the given server thread.

  Return type: bigint
- pg_stat_get_backend_dbid(integer)

  Description: Specifies the ID of the database connected to the given server process.

  Return type: oid
- pg_stat_get_backend_userid(integer)

  Description: Specifies the user ID of the given server process.

  Return type: oid
- pg_stat_get_backend_activity(integer)

  Description: Active command of the given server process, but only if the current user is a system administrator or the same user as that of the session being queried and **track_activities** is on

  Return type: text

- pg_stat_get_backend_waiting(integer)

  Description: True if the given server process is waiting for a lock, but only if the current user is a system administrator or the same user as that of the session being queried and **track_activities** is on

  Return type: Boolean
- pg_stat_get_backend_activity_start(integer)

  Description: Specifies the time when the given server process's currently executing query is started only if the current user is the system administrator or the user of the session being queried and **track_activities** is enabled.

  Return type: timestamp with time zone
- pg_stat_get_backend_xact_start(integer)

  Description: Specifies the time when the given server process's currently executing transaction is started only if the current user is the system administrator or the user of the session being queried and **track_activities** is enabled.

  Return type: timestamp with time zone
- pg_stat_get_backend_start(integer)

  Description: Specifies the time when the given server process is started. If the current user is neither the system administrator nor the user of the session being queried, NULL is **returned**.

  Return type: timestamp with time zone
- pg_stat_get_backend_client_addr(integer)

  Description: Specifies the IP address of the client connected to the given server process. If the connection is over a Unix domain socket, or if the current user is neither a system administrator nor the same user as that of the session being queried, **NULL** will be returned.

  Return type: inet
- pg_stat_get_backend_client_port(integer)

  Description: Specifies the TCP port number of the client connected to the given server process. If the connection is over a Unix domain socket, **-1** will be returned. If the current user is neither a system administrator nor the same user as that of the session being queried, **NULL** will be returned.

  Return type: integer
- pg_stat_get_bgwriter_timed_checkpoints()

  Description: Specifies the time when the background writer starts scheduled checkpoints (because the **checkpoint_timeout** time has expired).

  Return type: bigint
- pg_stat_get_bgwriter_requested_checkpoints()

  Description: Specifies the time when the background writer starts checkpoints based on requests from the backend because **checkpoint_segments** has been exceeded or the **CHECKPOINT** command has been executed.

  Return type: bigint
- pg_stat_get_bgwriter_buf_written_checkpoints()

  Description: Specifies the number of buffers written by the background writer during checkpoints.

Return type: bigint

- pg_stat_get_bgwriter_buf_written_clean()

  Description: Specifies the number of buffers written by the background writer for routine cleaning of dirty pages.

  Return type: bigint

- pg_stat_get_bgwriter_maxwritten_clean()

  Description: Specifies the time when the background writer stops its cleaning scan because it has written more buffers than specified in the **bgwriter_lru_maxpages** parameter.

  Return type: bigint

- pg_stat_get_buf_written_backend()

  Description: Specifies the number of buffers written by the backend because they need to allocate a new buffer.

  Return type: bigint

- pg_stat_get_buf_alloc()

  Description: Specifies the total number of the allocated buffers.

  Return type: bigint

- pg_stat_clear_snapshot()

  Description: Discards the current statistics snapshot.

  Return type: void

- pg_stat_reset()

  Description: Resets all statistics counters for the current database to zero (requires system administrator permissions).

  Return type: void

- pg_stat_reset_shared(text)

  Description: Resets all statistics counters for the current database in each node in a shared cluster to zero (requires system administrator permissions).

  Return type: void

- pg_stat_reset_single_table_counters(oid)

  Description: Resets statistics for a single table or index in the current database to zero (requires system administrator permissions).

  Return type: void

- pg_stat_reset_single_function_counters(oid)

  Description: Resets statistics for a single function in the current database to zero (requires system administrator permissions).

  Return type: void

- pg_stat_session_cu(int, int, int)

  Description: Obtains the compression unit (CU) hit statistics of sessions running on the current node.

  Return type: record

- pg_stat_get_cu_mem_hit(oid)

  Description: Obtains the number of CU memory hits of a column storage table in the current database of the current node.

Return type: bigint

- pg_stat_get_cu_hdd_sync(oid)

  Description: Obtains the times CU is synchronously read from a disk by a column storage table in the current database of the current node.

  Return type: bigint

- pg_stat_get_cu_hdd_asyn(oid)

  Description: Obtains the times CU is asynchronously read from a disk by a column storage table in the current database of the current node.

  Return type: bigint

- pg_stat_get_db_cu_mem_hit(oid)

  Description: Obtains the CU memory hit in a database of the current node.

  Return type: bigint

- pg_stat_get_db_cu_hdd_sync(oid)

  Description: Obtains the times CU is synchronously read from a disk by a database of the current node.

  Return type: bigint

- fenced_udf_process(integer)

  Description: Shows the number of local UDF Master and Work processes. If the input parameter is set to **1**, the number of Master processes is queried. If the input parameter is set to **2**, the number of Worker processes is queried. If the input parameter is set to **3**, all Worker processes are killed.

  Return type: text

- total_cpu()

  Description: Obtains the CPU time used by the current node, in jiffies.

  Return type: bigint

- total_memory()

  Description: Obtains the size of the virtual memory used by the current node, in KB.

  Return type: bigint

- pg_stat_get_db_cu_hdd_asyn(oid)

  Description: Obtains the times CU is asynchronously read from a disk by a database of the current node.

  Return type: bigint

- pg_stat_bad_block(text, int, int, int, int, int, timestamp with time zone, timestamp with time zone)

  Description: Obtains damage information about pages or CUs after the current node is started.

  Example: select * from pg_stat_bad_block();

  Return type: record

- pg_stat_bad_block_clear()

  Description: Deletes the page and CU damage information that is read and recorded on the node (requires system administrator permissions).

  Return type: void

- gs_respool_exception_info(pool text)

  Description: Queries the query rule of a specified resource pool.

  Return type: record

- gs_control_group_info(pool text)

  Description: Queries information about Cgroups associated with a resource pool.

  Return type: record

  The command output is as follows:

Attribute	Value	Description
name	class_a:workload_a1	Class name and workload name
class	class_a	Class Cgroup name
workload	workload_a1	Workload Cgroup name
type	DEFWD	Cgroup type (Top, CLASS, BAKWD, DEFWD, and TSWD)
gid	87	Cgroup ID
shares	30	Percentage of CPU resources to those on the parent node
limits	0	Percentage of CPU cores to those on the parent node
rate	0	Allocation ratio in Timeshare
cpucores	0-3	Number of CPU cores

- gs_all_control_group_info()

  Description: Collects information about all Cgroups in the database.

  Return type: record

- gs_get_control_group_info()

  Description: Collects information about all Cgroups.

  Return type: record

- get_instr_workload_info(integer)

  Description: Obtains the transaction volume and time information on the primary database node.

  Return type: record

Attribute	Value	Description
resourcepool_oid	10	OID of the resource pool (the logic is equivalent to the load)
commit_counter	4	Number of frontend transactions that were committed

Attribute	Value	Description
rollback_counter	1	Number of frontend transactions that were rolled back
resp_min	949	Minimum response time of frontend transactions (unit: µs)
resp_max	201891	Maximum response time of frontend transactions (unit: µs)
resp_avg	43564	Average response time of frontend transactions (unit: µs)
resp_total	217822	Total response time of frontend transactions (unit: µs)
bg_commit_counter	910	Number of backend transactions that were committed
bg_rollback_counter	0	Number of backend transactions that were rolled back
bg_resp_min	97	Minimum response time of backend transactions (unit: µs)
bg_resp_max	678080687	Maximum response time of backend transactions (unit: µs)
bg_resp_avg	327847884	Average response time of backend transactions (unit: µs)
bg_resp_total	298341575300	Total response time of backend transactions (unit: µs)

- pv_instance_time()

  Description: Obtains the time consumed in each execution phase on the current node.

  Return type: record

Stat_name Attribute	Value	Description
DB_TIME	1062385	Total end-to-end wall time consumed by all threads (unit: µs)
CPU_TIME	311777	Total CPU time consumed by all threads (unit: µs)
EXECUTION_TIME	380037	Total time consumed on the executor (unit: µs)
PARSE_TIME	6033	Total time consumed for parsing SQL statements (unit: µs)

Stat_name Attribute	Value	Description
PLAN_TIME	173356	Total time consumed for generating an execution plan (unit: μs)
REWRITE_TIME	2274	Total time consumed on query rewriting (unit: μs)
PL_EXECUTION_TIME	0	Total time consumed for executing PL/SQL statements (unit: μs)
PL_COMPILATION_TIME	557	Total time consumed for SQL compilation (unit: μs)
NET_SEND_TIME	1673	Total time consumed for sending data over network (unit: μs)
DATA_IO_TIME	426622	Total time consumed for data read and write (unit: μs)

- DBE_PERF.get_global_instance_time()

  Description: Provides the time consumed in each key phase in the entire database. To query this function, you must have the sysadmin permission.

  Return type: record

- get_instr_unique_sql()

  Description: Obtains information about execution statements (normalized SQL statements) on the current node as a user with the sysadmin permission.

  Return type: record

- reset_unique_sql(text, text, bigint)

  Description: Resets information about system execution statements (normalized SQL statements) information as a user with the sysadmin permission. The value of the first parameter can be **global** or **local**. **global** indicates that information on all nodes is cleared, and **local** indicates that only information on the current node is cleared. The value of the second parameter can be **ALL**, **BY_USERID**, or **BY_CNID**. **ALL** indicates that all information is cleared. **BY_USERID** indicates that the SQL information of the user specified by **USERID** is cleared. **BY_CNID** indicates that the SQL information related to the primary node of the database in the system is cleared. The third parameter indicates **CNID** and **USERID**. If the second parameter is set to **ALL**, the third parameter does not take effect and can be set to any value.

  Return type: Boolean

  NOTE

  This function involves distributed nodes. Currently, GaussDB is a centralized database, for which the function of the value **global** is the same as that of the value **local** and the second parameter cannot set to be **BY_CNID**.

- get_instr_wait_event(NULL)

  Description: Obtains the statistics on wait events of the current node.

Return type: record

- get_instr_user_login()

  Description: Obtains the number of user login and logout times on the current node. Only users with the sysadmin or monitor admin permission can execute this function.

  Return type: record

- get_instr_rt_percentile(integer)

  Description: Obtains the SQL response time P80 and P95 distribution information of the database.

  Return type: record

- get_node_stat_reset_time()

  Description: Obtains statistics about reset (restart, primary/standby switchover, and database deletion) time of the current node.

  Return type: record

- DBE_PERF.get_global_os_runtime()

  Description: Displays the running status of the current operating system. To query this function, you must have the sysadmin permission.

  Return type: record

- DBE_PERF.get_global_os_threads()

  Description: Provides information about the threads under all normal nodes of the entire database. To query this function, you must have the sysadmin permission.

  Return type: record

- DBE_PERF.get_summary_workload_sql_count()

  Description: Provides statistics about the number of SELECT, UPDATE, INSERT, DELETE, DDL, DML, and DCL statements of different service loads in the entire database. To query this function, you must have the sysadmin permission.

  Return type: record

- DBE_PERF.get_summary_workload_sql_elapse_time()

  Description: Provides statistics about the number of SELECT, UPDATE, INSERT, and DELETE statements and response time information (TOTAL, AVG, MIN, and MAX) for different loads in the entire database. To query this function, you must have the sysadmin permission.

  Return type: record

- DBE_PERF.get_global_workload_transaction()

  Description: Obtains the transaction volume and time information on all nodes of the database. To query this function, you must have the sysadmin permission.

  Return type: record

- DBE_PERF.get_global_session_stat()

  Description: Obtains the session status information on all nodes of the database. To query this function, you must have the sysadmin permission.

  Return type: record

NOTE

The status information contains the following 17 items: **commit**, **rollback**, **sql**, **table_scan**, **blocks_fetched**, **physical_read_operation**, **shared_blocks_dirtied**, **local_blocks_dirtied**, **shared_blocks_read**, **local_blocks_read**, **blocks_read_time**, **blocks_write_time**, **sort_imemory**, **sort_idisk**, **cu_mem_hit**, **cu_hdd_sync_read**, and **cu_hdd_asyread**.

- DBE_PERF.get_global_session_time()

  Description: Provides the time consumed in each key phase of each node in the entire database. To query this function, you must have the sysadmin permission.

  Return type: record

- DBE_PERF.get_global_session_memory()

  Description: Displays statistics about memory usage at the session level on each node in the unit of MB, including all the memory allocated to Postgres and stream threads on DNs for jobs currently executed by users. To query this function, you must have the sysadmin permission.

  Return type: record

- DBE_PERF.get_global_session_memory_detail()

  Description: Displays statistics about thread memory usage on each node by MemoryContext node. To query this function, you must have the sysadmin permission.

  Return type: record

- create_wlm_session_info(int flag)

  Description: Clears top SQL query statement-level statistics recorded in the current memory. Only the administrator can execute this function.

  Return type: int

- pg_stat_get_wlm_session_info(int flag)

  Description: Obtains top SQL query statement-level statistics recorded in the current memory. If the input parameter is not 0, the information is cleared from the memory. Only users with the system admin or monitor admin permission can execute this function.

  Return type: record

- gs_paxos_stat_replication()

  Description: Queries the standby node information on the primary node. Currently, only the centralized DCF mode is supported.

  Return type: setofrecord

  The following table describes return columns.

Column	Type	Description
local_role	text	Role of the node that sends logs
peer_role	text	Role of the node that receives logs
local_dcf_role	text	DCF role of the node that sends logs
peer_dcf_role	text	DCF role of the node that receives logs

peer_state	text	Status of the node that receives logs
sender_write_location	text	Location in the Xlog buffer where the node that sends logs is written
sender_commit_location	text	Consistency point reached for the DCF logs of the node that sends logs
sender_flush_location	text	Location in the Xlog disk where the node that sends logs is written
sender_replay_location	text	Location where the node that sends logs replays logs
receiver_write_location	text	Location in the Xlog buffer where the node that receives logs is written
receiver_commit_location	text	Consistency point reached for the DCF logs of the node that receives logs
receiver_flush_location	text	Location in the Xlog disk where the node that receives logs is written
receiver_replay_location	text	Location where the node that receives logs replays Xlogs
sync_percent	text	Synchronization percentage
dcf_run_mode	int4	DCF synchronization mode
channel	text	Channel information

- gs_wlm_get_resource_pool_info(int)

  Description: Obtains resource usage statistics of all users. The input parameter can be any value of the INT type or be **NULL**.

  Return type: record
- gs_wlm_get_all_user_resource_info()

  Description: Obtains resource usage statistics of all users.

  Return type: record
- gs_wlm_get_user_info(int)

  Description: Obtains information about all users. The input parameter is of the int type and can be any int value or **NULL**. Only users with the sysadmin permission can execute this function.

  Return type: record
- gs_wlm_get_workload_records()

  Description: Obtains all job information in dynamic load management. This function is valid only when dynamic load management is enabled. (The current feature is a lab feature. Contact Huawei technical support before using it.)

  Return type: record
- gs_wlm_readjust_user_space()

Description: Corrects the storage space usage of all users. Only the administrator can execute this function.

Return type: record

- gs_wlm_readjust_user_space_through_username(text name)

  Description: Corrects the storage space usage of a specified user. Common users can use this function to modify only their own usage. Only the administrator can modify the usage of all users. If the value of **name** is **0000**, the usage of all users needs to be modified.

  Return type: record

- gs_wlm_readjust_user_space_with_reset_flag(text name, boolean isfirst)

  Description: Corrects the storage space usage of a specified user. If the input parameter **isfirst** is set to **true**, statistics are collected from 0. Otherwise, statistics are collected from the previous result. Common users can use this function to modify only their own usage. Only the administrator can modify the usage of all users. If the value of **name** is **0000**, the usage of all users needs to be modified.

  Return type: record

- gs_wlm_session_respool(bigint)

  Description: Obtains the session resource pool information about all backend threads. The input parameter can be any value of the bigint type or can be null.

  Return type: record

- gs_wlm_get_session_info()

  Description: This API has been discarded and is unavailable currently.

- gs_wlm_get_user_session_info()

  Description: This API has been discarded and is unavailable currently.

- gs_io_wait_status()

  Description: This API does not support single-node systems or centralized systems and is unavailable currently.

- global_stat_get_hotkeys_info()

  Description: Obtains the statistics on hot keys in the entire database instance. This API does not support single-node systems or centralized systems and is unavailable currently.

- global_stat_clean_hotkeys()

  Description: Clears statistics on hot keys in the entire database instance. This API does not support single-node systems or centralized systems and is unavailable currently.

- DBE_PERF.get_global_session_stat_activity()

  Description: Displays information about threads that are running on each node in the database. To query this function, you must have the monitoradmin permission.

  Return type: record

- DBE_PERF.get_global_thread_wait_status()

  Description: Displays the block waiting status of backend threads and auxiliary threads on all nodes. To query this function, you must have the sysadmin or monitoradmin permission.

Return type: record

- DBE_PERF.get_global_operator_history_table()

  Description: Displays the operator-related records (persistent) generated after jobs are executed on the primary database node of the current user. To query this function, you must have the sysadmin or monitoradmin permission.

  Return type: record

- DBE_PERF.get_global_operator_history()

  Description: Displays the operator-related records generated after jobs are executed on the primary database node of the current user. To query this function, you must have the sysadmin or monitoradmin permission.

  Return type: record

- DBE_PERF.get_global_operator_runtime()

  Description: Displays real-time operator-related records of jobs executed on the primary database node of the current user. To query this function, you must have the sysadmin or monitoradmin permission.

  Return type: record

- DBE_PERF.get_global_statement_complex_history()

  Description: Displays the historical records of complex queries on the primary database node of the current user. To query this function, you must have the monitoradmin permission.

  Return type: record

- DBE_PERF.get_global_statement_complex_history_table()

  Description: Displays the historical records (persistent) of complex queries on the primary database node of the current user. To query this function, you must have the monitoradmin permission.

  Return type: record

- DBE_PERF.get_global_statement_complex_runtime()

  Description: Displays the real-time information of complex queries on the primary database node of the current user. To query this function, you must have the sysadmin or monitoradmin permission.

  Return type: record

- DBE_PERF.get_global_memory_node_detail()

  Description: Displays the memory usage of a certain database on all nodes. To query this function, you must have the monitoradmin permission.

  Return type: record

- DBE_PERF.get_global_shared_memory_detail()

  Description: Displays the usage information about all the shared memory contexts of all nodes. To query this function, you must have the monitoradmin permission.

  Return type: record

- DBE_PERF.get_global_statio_all_indexes()

  Description: Displays statistics about each index displayed in a row in the current database, showing I/O statistics about accesses to that specific index. To query this function, you must have the sysadmin permission.

  Return type: record

- DBE_PERF.get_summary_stat_all_tables()

  Description: Displays statistics about a row in each table (including the TOAST table) on each node.

  Return type: record
- DBE_PERF.get_global_stat_all_tables()

  Description: Displays statistics about a row in each table (including the TOAST table) on each node.

  Return type: record
- DBE_PERF.get_local_toastname_and_toastindexname()

  Description: Provides the mapping between the name and index of the local TOAST table and its associated table. To query this function, you must have the sysadmin permission.

  Return type: record
- DBE_PERF.get_summary_statio_all_indexes()

  Description: Collects statistics about each index displayed in a row in the current databases of all nodes and displays the I/O statistics of a specific index. To query this function, you must have the sysadmin permission.

  Return type: record
- DBE_PERF.get_global_statio_all_sequences()

  Description: Provides I/O status information about all sequences in the namespace. To query this function, you must have the sysadmin permission.

  Return type: record
- DBE_PERF.get_global_statio_all_tables()

  Description: Displays the I/O statistics about each table in databases on each node. To query this function, you must have the sysadmin permission.

  Return type: record
- DBE_PERF.get_summary_statio_all_tables()

  Description: Collects I/O statistics about each table in databases in the database. To query this function, you must have the sysadmin permission.

  Return type: record
- DBE_PERF.get_local_toast_relation()

  Description: Provides the mapping between the name of the local TOAST table and its associated table. To query this function, you must have the sysadmin permission.

  Return type: record
- DBE_PERF.get_global_statio_sys_indexes()

  Description: Displays the I/O status information about all system catalog indexes in namespaces on each node. To query this function, you must have the sysadmin permission.

  Return type: record
- DBE_PERF.get_summary_statio_sys_indexes()

  Description: Collects the I/O status information about all system catalog indexes in namespaces on each node. To query this function, you must have the sysadmin permission.

Return type: record

- DBE_PERF.get_global_statio_sys_sequences()

  Description: Provides the I/O status information about all the system sequences in the namespace. To query this function, you must have the sysadmin permission.

  Return type: record

- DBE_PERF.get_global_statio_sys_tables()

  Description: Provides I/O status information about all system catalogs in namespaces on each node. To query this function, you must have the sysadmin permission.

  Return type: record

- DBE_PERF.get_summary_statio_sys_tables()

  Description: Displays the I/O status information of all system catalogs in the namespace in the database. To query this function, you must have the sysadmin permission.

  Return type: record

- DBE_PERF.get_global_statio_user_indexes()

  Description: Displays the I/O status information about all user relationship table indexes in namespaces on each node. To query this function, you must have the sysadmin permission.

  Return type: record

- DBE_PERF.get_summary_statio_user_indexes()

  Description: Displays the I/O status information about all user relationship table indexes in namespaces in the database. To query this function, you must have the sysadmin permission.

  Return type: record

- DBE_PERF.get_global_statio_user_sequences()

  Description: Displays the I/O status information about all user sequences in the namespace of each node. To query this function, you must have the sysadmin permission.

  Return type: record

- DBE_PERF.get_global_statio_user_tables()

  Description: Displays the I/O status information about all user relationship tables in namespaces on each node. To query this function, you must have the sysadmin permission.

  Return type: record

- DBE_PERF.get_summary_statio_user_tables()

  Description: Displays the I/O status information about all user relationship tables in namespaces in the database. To query this function, you must have the sysadmin permission.

  Return type: record

- DBE_PERF.get_stat_db_cu()

  Description: Queries CU hits in a database and in each node in the database. To query this function, you must have the sysadmin permission.

  Return type: record

- DBE_PERF.get_global_stat_all_indexes()

  Description: Displays statistics of each index in databases on all nodes. To query this function, you must have the sysadmin permission.

  Return type: record
- DBE_PERF.get_summary_stat_all_indexes()

  Description: Collects statistics of each index in all databases on all nodes. To query this function, you must have the sysadmin permission.

  Return type: record
- DBE_PERF.get_global_stat_sys_tables()

  Description: Displays statistics about the system catalogs of all the namespaces in **pg_catalog** and **information_schema** schemas on each node. To query this function, you must have the sysadmin permission.

  Return type: record
- DBE_PERF.get_summary_stat_sys_tables()

  Description: Collects statistics about the system catalogs of all the namespaces in **pg_catalog** and **information_schema** schemas on each node. To query this function, you must have the sysadmin permission.

  Return type: record
- DBE_PERF.get_global_stat_sys_indexes()

  Description: Displays index status information about all the system catalogs in the **pg_catalog** and **information_schema** schemas on each node. To query this function, you must have the sysadmin permission.

  Return type: record
- DBE_PERF.get_summary_stat_sys_indexes()

  Description: Collects statistics about index status information about all the system catalogs in the **pg_catalog** and **information_schema** schemas on each node. To query this function, you must have the sysadmin permission.

  Return type: record
- DBE_PERF.get_global_stat_user_tables()

  Description: Displays the status information about customized ordinary tables in all namespaces. To query this function, you must have the sysadmin permission.

  Return type: record
- DBE_PERF.get_summary_stat_user_tables()

  Description: Collects statistics about the status information about customized ordinary tables in all namespaces. To query this function, you must have the sysadmin permission.

  Return type: record
- DBE_PERF.get_global_stat_user_indexes()

  Description: Displays the status information about the index of customized ordinary tables in all databases. To query this function, you must have the sysadmin permission.

  Return type: record
- DBE_PERF.get_summary_stat_user_indexes()

Description: Collects statistics about the status information about the index of customized ordinary tables in all databases. To query this function, you must have the sysadmin permission.

Return type: record

- DBE_PERF.get_global_stat_database()

  Description: Displays database statistics of all nodes. To query this function, you must have the sysadmin permission.

  Return type: record

- DBE_PERF.get_global_stat_database_conflicts()

  Description: Collects statistics on the database of all nodes. To query this function, you must have the sysadmin permission.

  Return type: record

- DBE_PERF.get_global_stat_xact_all_tables()

  Description: Displays transaction status information about all ordinary tables and TOAST tables in all namespaces. To query this function, you must have the sysadmin permission.

  Return type: record

- DBE_PERF.get_summary_stat_xact_all_tables()

  Description: Collects statistics about transaction status information about all ordinary tables and TOAST tables in all namespaces. To query this function, you must have the sysadmin permission.

  Return type: record

- DBE_PERF.get_global_stat_xact_sys_tables()

  Description: Displays transaction status information about all system catalogs in namespaces on all nodes. To query this function, you must have the sysadmin permission.

  Return type: record

- DBE_PERF.get_summary_stat_xact_sys_tables()

  Description: Collects statistics about transaction status information about all system catalogs in namespaces on all nodes. To query this function, you must have the sysadmin permission.

  Return type: record

- DBE_PERF.get_global_stat_xact_user_tables()

  Description: Displays the transaction status information of the user tables in the namespaces on all nodes. To query this function, you must have the sysadmin permission.

  Return type: record

- DBE_PERF.get_summary_stat_xact_user_tables()

  Description: Collects statistics about the transaction status information of the user tables in the namespaces on all nodes. To query this function, you must have the sysadmin permission.

  Return type: record

- DBE_PERF.get_global_stat_user_functions()

Description: Displays the transaction status information of customized functions in the namespaces on all nodes. To query this function, you must have the sysadmin permission.

Return type: record

- DBE_PERF.get_global_stat_xact_user_functions()

  Description: Collects statistics about the transaction status information of customized functions in the namespaces on all nodes. To query this function, you must have the sysadmin permission.

  Return type: record

- DBE_PERF.get_global_stat_bad_block()

  Description: Displays information about table and index read failures on all nodes. To query this function, you must have the sysadmin permission.

  Return type: record

- DBE_PERF.get_global_file_redo_iostat()

  Description: Collects statistics on information about table and index read failures on all nodes. To query this function, you must have the sysadmin permission.

  Return type: record

- DBE_PERF.get_global_file_iostat()

  Description: Displays statistics about data file I/Os on all nodes. To query this function, you must have the sysadmin permission.

  Return type: record

- DBE_PERF.get_global_locks()

  Description: Displays lock information of all nodes. To query this function, you must have the sysadmin or monadmin permission.

  Return type: record

- DBE_PERF.get_global_replication_slots()

  Description: Displays logical replication information on all nodes. To query this function, you must have the sysadmin or monadmin permission.

  Return type: record

- DBE_PERF.get_global_bgwriter_stat()

  Description: Displays statistics about the background writer process's activities on all nodes. To query this function, you must have the sysadmin permission.

  Return type: record

- DBE_PERF.get_global_replication_stat()

  Description: Displays information about log synchronization status on each node, such as the locations where the sender sends logs and where the receiver receives logs. To query this function, you must have the sysadmin or monadmin permission.

  Return type: record

- DBE_PERF.get_global_transactions_running_xacts()

  Description: Displays information about running transactions on each node. To query this function, you must have the sysadmin permission.

  Return type: record

- DBE_PERF.get_summary_transactions_running_xacts()

  Description: Collects statistics of information about running transactions on each node. To query this function, you must have the sysadmin permission.

  Return type: record

- DBE_PERF.get_global_transactions_prepared_xacts()

  Description: Displays information about transactions that are currently prepared for two-phase commit on each node. To query this function, you must have the sysadmin permission.

  Return type: record

- DBE_PERF.get_summary_transactions_prepared_xacts()

  Description: Collects statistics information about transactions that are currently prepared for two-phase commit on each node. To query this function, you must have the sysadmin permission.

  Return type: record

- DBE_PERF.get_summary_statement()

  Description: Displays the status information of the historically-executed statements on each node. To query this function, you must have the sysadmin or monitor admin permission.

  Return type: record

- DBE_PERF.get_global_statement_count()

  Description: Displays the number of SELECT, UPDATE, INSERT, and DELETE statements and response time information (TOTAL, AVG, MIN, and MAX) on each node. To query this function, you must have the sysadmin permission.

  Return type: record

- DBE_PERF.get_global_config_settings()

  Description: Displays GUC parameter configuration information on each node. To query this function, you must have the sysadmin permission.

  Return type: record

- DBE_PERF.get_global_wait_events()

  Description: Displays the wait event status information on each node. To query this function, you must have the sysadmin permission.

  Return type: record

- DBE_PERF.get_statement_responsetime_percentile()

  Description: Obtains the response time distribution for 80% and 95% SQL statements of the database. To query this function, you must have the sysadmin permission.

  Return type: record

- DBE_PERF.get_summary_user_login()

  Description: Collects statistics about number of user login and logout times on each node in the database. To query this function, you must have the sysadmin permission.

  Return type: record

- DBE_PERF.get_global_record_reset_time()

Description: Displays the statistics about reset (restart, primary/standby switchover, and database deletion) time of the database. To query this function, you must have the sysadmin permission.

Return type: record

- DBE_PERF.track_memory_context(context_list text)

  Description: Sets the memory context whose memory application details need to be collected. The input parameter is the memory context names, which are separated by commas (,), for example, **ThreadTopMemoryContext, SessionCacheMemoryContext**. Note that the memory context names are context-sensitive. In addition, the length of a single memory context is 63, and the excess part is truncated. The maximum number of memory contexts that can be collected at a time is 16. If the number of memory contexts exceeds 16, the setting fails. Each time this function is called, the previous statistics result is cleared. When the input parameter is set to **""**, the statistics function is disabled. Only the initial user or a user with the monadmin permission can execute this function.

  Return type: Boolean

- DBE_PERF.track_memory_context_detail()

  Description: Obtains the memory application details of the memory context specified by the **DBE_PERF.track_memory_context** function. For details, see the **DBE_PERF.track_memory_context_detail** view. Only the initial user or a user with the monadmin permission can execute this function.

  Return type: record

- pg_stat_get_mem_mbytes_reserved(tid)

  Description: Collects statistics on variables related to resource management, which is used only for fault locating.

  Parameter: thread ID

  Return type: text

- gs_wlm_user_resource_info(name text)

  Description: Queries a user's resource quota and resource usage.

  Return type: record

- pg_stat_get_file_stat()

  Description: Records statistics about data file I/Os to indicate I/O performance and detect performance problems such as abnormal I/O operations.

  Return type: record

- pg_stat_get_redo_stat()

  Description: Displays statistics on the replay of session thread logs.

  Return type: record

- pg_stat_get_status(int8)

  Description: Tests the block waiting status about the backend thread and auxiliary thread of the current instance.

  Return type: record

- get_local_rel_iostat()

  Description: Queries the accumulated I/O status of data files on the current node.

Return type: record

- DBE_PERF.get_global_rel_iostat()

  Description: Displays statistics about data file I/Os on all nodes. To query this function, you must have the sysadmin permission.

  Return type: record

- DBE_PERF.global_threadpool_status()

  Description: Displays the status of worker threads and sessions in thread pools on all nodes. For details about the columns returned by the function, see **GLOBAL_THREADPOOL_STATUS**.

  Return type: record

- remote_bgwriter_stat()

  Description: Displays the information about pages flushed by the bgwriter threads of all instances in the database, number of pages in the candidate buffer chain, and buffer elimination information (except for the local node and not available on the DN).

  Return type: record

- pv_os_run_info

  Description: Displays the running status of the current OS. For details about the columns, see **GS_OS_RUN_INFO**.

  Parameter: nan

  Return type: SETOF record

- pv_session_stat

  Description: Collects session status information by session thread or AutoVacuum thread. For details about the columns, see **GS_SESSION_STAT**.

  Parameter: **nan**

  Return type: SETOF record

- pv_session_time

  Description: Collects statistics on the running time of session threads and the time consumed in each execution phase. For details about the columns, see **GS_SESSION_TIME**.

  Parameter: **nan**

  Return type: SETOF record

- pg_stat_get_db_temp_bytes

  Description: Collects statistics on the total amount of data written to temporary files through database query. All temporary files are counted, regardless of why the temporary file was created, and regardless of the **log_temp_files** setting.

  Parameter: **oid**

  Return type: bigint

- pg_stat_get_db_temp_files

  Description: Queries the number of temporary files created in the database. All temporary files are counted, regardless of why the temporary file was created (for example, sorting or hashing), and regardless of the **log_temp_files** setting.

  Parameter: **oid**

Return type: bigint

- create_wlm_instance_statistics_info

  Description: Saves the historical monitoring data of the current instance persistently.

  Parameter: **nan**

  Return type: integer

Table 11-84 remote_bgwriter_stat parameter description

Parameter	Type	Description
node_name	text	Instance name
bgwr_actual_flush_total_num	bigint	Total number of dirty pages flushed by the bgwriter thread from the startup time to the current time
bgwr_last_flush_num	integer	Number of dirty pages flushed by the bgwriter thread in the previous batch
candidate_slots	integer	Number of pages in the current candidate buffer chain
get_buffer_from_list	bigint	Number of times that pages are obtained from the candidate buffer chain during buffer eviction
get_buf_clock_sweep	bigint	Number of times that pages are obtained from the original eviction solution during buffer eviction

- remote_candidate_stat()

  Description: Displays the checkpoint information and log flushing information about all instances in the database (except the current node). Centralized systems are not supported.

  Return type: record

- remote_ckpt_stat()

  Description: Displays the checkpoint information and log flushing information about all instances in the database (except the current node). Centralized systems are not supported.

  Return type: record

- remote_single_flush_dw_stat()

  Description: Displays the single-page doublewrite file status of all instances in the database (except the current node). Centralized systems are not supported.

  Return type: record

- remote_double_write_stat()

Description: Displays doublewrite file status of all instances in the database (except the current node). Centralized systems are not supported.

Return type: record

- remote_pagewriter_stat()

  Description: Displays the page flushing information and checkpoint information about all instances in the database (except the current node). Centralized systems are not supported.

  Return type: record

- remote_recovery_status()

  Description: Displays log flow control information about the primary and standby nodes (except the current node). Centralized systems are not supported.

  Return type: record

- remote_redo_stat()

  Description: Displays the log replay status of all instances in the database (except the current node). Centralized systems are not supported.

  Return type: record

  Example:

  The function **pg_backend_pid** shows the ID of the current server thread.

```
openGauss=# SELECT pg_backend_pid();
 pg_backend_pid
-----------------
 139706243217168
(1 row)
```

  The function **pg_stat_get_backend_pid** shows the ID of a given server thread.

```
openGauss=# SELECT pg_stat_get_backend_pid(1);
 pg_stat_get_backend_pid
-------------------------
         139706243217168
(1 row)
```

- dbe_perf.gs_stat_activity_timeout(int)

  Description: Obtains information about query jobs whose execution time exceeds the timeout threshold on the current node. The correct result can be returned only when the GUC parameter **track_activities** is set to **on**. The timeout threshold ranges from 0 to 2147483.

  Return type: SETOF record

Name	Type	Description
database	name	Name of the database to which a user session is connected
pid	bigint	Backend thread ID
sessionid	bigint	Session ID
usesysid	oid	OID of the user logged in to the backend
application_name	text	Name of the application connected to the backend

Name	Type	Description
query	text	Query that is being executed on the backend
xact_start	timestamptz	Time when the current transaction is started
query_start	timestamptz	Time when the current query starts
query_id	bigint	Query statement ID

- gs_wlm_user_resource_info(name text)

  Description: Queries a user's resource quota and resource usage. Common users can query only their own information. Administrators can query information about all users.

  Return type: record
- create_wlm_instance_statistics_info

  Description: Saves the historical monitoring data of the current instance persistently.

  Parameter: **nan**

  Return type: integer
- gs_session_memory

  Description: Collects statistics about memory usage at the session level, including all the memory allocated to Postgres and Stream threads on DNs for tasks currently executed by users.

  NOTE

  If **enable_memory_limit** is set to **off**, this function cannot be used.

  Return type: record

Table 11-85 Return value description

Name	Type	Description
sessid	text	Thread start time and ID
init_mem	integer	Memory allocated to the currently executed jobs before they enter the executor, in MB
used_mem	integer	Memory allocated to the currently executed jobs, in MB
peak_mem	integer	Peak memory allocated to the currently executed jobs, in MB

- gs_wlm_persistent_user_resource_info()

  Description: Archives all user resource usage statistics to the **gs_wlm_user_resource_history** system catalog. To query this function, you must have the sysadmin permission.

Return type: record

- create_wlm_operator_info(int flag)

  Description: Clears top SQL operator-level statistics recorded in the current memory. If the input parameter is greater than 0, the information is archived to **gs_wlm_operator_info** and **gs_wlm_ec_operator_info**. Otherwise, the information is not archived. Only users with the sysadmin permission can execute this function.

  Return type: int

- GS_ALL_NODEGROUP_CONTROL_GROUP_INFO(text)

  Description: Provides Cgroup information for all logical database instances. Before calling this function, you need to specify the name of the logical database instance to be queried. For example, to query the Cgroup information for the installation logical database instance, run the following command:

  ```
  SELECT * FROM GS_ALL_NODEGROUP_CONTROL_GROUP_INFO('installation')
  ```

  Return type: record

  The following table describes return columns.

Name	Type	Description
name	text	Cgroup name.
type	text	Cgroup type.
gid	bigint	Cgroup ID.
classgid	bigint	ID of the class Cgroup where a workload Cgroup belongs.
class	text	Class Cgroup.
workload	text	Workload Cgroup.
shares	bigint	CPU quota allocated to the Cgroup.
limits	bigint	Limit of CPU resources allocated to a Cgroup.
wdlevel	bigint	Workload Cgroup level.
cpucores	text	Information about the CPU cores used by a Cgroup.

- gs_total_nodegroup_memory_detail

  Description: Returns information about the memory used by the current logical database, in MB.

  Return type: SETOF record

- local_redo_time_count()

  Description: Returns the time consumption statistics on each process of each playback thread on the current node (valid data exists only on the standby node).

  The return values are as follows:

  local_redo_time_count parameters

Column	Description
thread_name	Thread name
step1_total	Total duration of step 1. The process of each thread is as follows: Ultimate RTO ● **batch redo**: obtains a log from a queue. ● **redo manager**: obtains a log from a queue. ● **redo worker**: obtains a log from a queue. ● **trxn manager**: reads a log from a queue. ● **trxn worker**: reads a log from a queue. ● **read worker**: reads an Xlog page (overall) from a file. ● **read page worker**: obtains a log from a queue. ● **startup**: obtains a log from a queue. Parallel replay: ● **page redo**: obtains a log from a queue. ● **startup**: reads a log.
step1_count	Number of accumulated execution times of step 1.
step2_total	Total duration of step 2. The process of each thread is as follows: Ultimate RTO ● **batch redo**: processes logs (overall). ● **redo manager**: processes logs (overall). ● **redo worker**: processes logs (overall). ● **trxn manager**: processes logs (overall). ● **trxn worker**: processes logs (overall). ● **redo worker**: specifies the time required for reading the Xlog page. ● **read page worker**: generates and sends LSN forwarders. ● **startup**: checks whether to replay to the specified position. Parallel replay: ● **page redo**: processes logs (overall). ● **startup**: checks whether to replay to the specified position.
step2_count	Number of accumulated execution times of step 2.

Column	Description
step3_total	Total duration of step 3. The process of each thread is as follows: Ultimate RTO ● **batch redo**: updates the standby state. ● **redo manager**: processes data logs. ● **redo worker**: replays page logs (overall). ● **trxn manager**: updates the flush LSN. ● **trxn worker**: replays logs. ● **redo worker**: pushes the Xlog segment. ● **read page worker**: obtains a new item. ● **startup**: collects statistics on the wait time of delayed replay feature. Parallel replay: ● **page redo**: updates the standby state. ● **startup**: collects statistics on the wait time of delayed replay feature.
step3_count	Number of accumulated execution times of step 3.
step4_total	Total duration of step 4. The process of each thread is as follows: Ultimate RTO: ● **batch redo**: parses Xlogs. ● **redo manager**: processes DDL. ● **redo worker**: reads data pages. ● **trxn manager**: synchronizes the wait time. ● **trxn worker**: updates the LSN of the current thread. ● **read page worker**: stores logs in the distribution thread. ● **startup**: distributes logs (overall). Parallel replay: ● **page redo**: replays undo logs. ● **startup**: distributes logs (overall).
step4_count	Number of accumulated execution times of step 4.

Column	Description
step5_total	Total duration of step 5. The process of each thread is as follows: Ultimate RTO: ● **batch redo**: distributes logs to the redo manager. ● **redo manager**: distributes logs to redo workers. ● **redo worker**: replays data page logs. ● **trxn manager**: distributes data to the trxn worker. ● **trxn worker**: forcibly synchronizes the wait time. ● **read page worker**: updates the LSN of the current thread. ● **startup**: decodes logs. Parallel replay: ● **page redo**: replays sharetrxn logs. ● **startup**: replays logs.
step5_count	Number of accumulated execution times of step 5.
step6_total	Total duration of step 6. The process of each thread is as follows: Ultimate RTO: ● **redo worker**: replays non-data page logs. ● **trxn manager**: updates global LSNs. ● **read page worker**: performs log CRC check. Parallel replay: ● **page redo**: replays synctrxn logs. ● **startup**: forcibly synchronizes the wait time.
step6_count	Number of accumulated execution times of step 6.
step7_total	Total duration of step 7. The process of each thread is as follows: Ultimate RTO: ● **redo manager**: creates tablespaces. ● **redo worker**: updates FSM. Parallel replay: **page redo**: replays a single log.

Column	Description
step7_count	Number of accumulated execution times of step 7.
step8_total	Total duration of step 8. The process of each thread is as follows: Ultimate RTO: **redo worker**: forcibly synchronizes the wait time. Parallel replay: **page redo**: replays all workers do log.
step8_count	Number of accumulated execution times of step 8.
step9_total	Total duration of step 9. The process of each thread is as follows: Ultimate RTO: None Parallel replay: **page redo**: replays muliti workers do log.
step9_count	Number of accumulated execution times of step 9.

- local_xlog_redo_statics()

  Description: Returns the statistics on each type of logs that have been replayed on the current node (valid data exists only on the standby node).

  The return values are as follows:

  Table 11-86 local_xlog_redo_statics parameters

Column	Description
xlog_type	Log types.
rmid	resource manager id
info	xlog operation
num	Number of logs.
extra	Valid values are available for page replay logs and xact logs. The page replay log indicates the number of pages read from the disk. The xact log indicates the number of deleted files.

- gs_get_shared_memctx_detail(text)

  Description: Returns the memory application details of the specified memory context, including the file, line number, and size of each memory application (the size of the same line in the same file is accumulated). Only the memory context queried through the **pg_shared_memory_detail** view can be queried. The input parameter is the memory context name (that is, the **contextname** column in the result returned by the **pg_shared_memory_detail** view). To query this function, you must have the sysadmin or monitor admin permission.

Return type: SETOF record

Name	Type	Description
file	text	Name of the file where the memory is applied for.
line	int8	Line number of the code in the file where the requested memory is located.
size	int8	Size of the applied memory. The value is accumulated if the memory is applied for multiple times in the same line of the same file.

NOTE

This view is not supported in the Lite release version.

- gs_get_session_memctx_detail(text)

  Description: Returns the memory application details of the specified memory context, including the file, line number, and size of each memory application (the size of the same line in the same file is accumulated). This parameter is valid only in thread pool mode. Only the memory context queried through the **gs_session_memory_context** view can be queried. The input parameter is the memory context name (that is, the **contextname** column in the result returned by the **gs_session_memory_context** view). To query this function, you must have the sysadmin or monitor admin permission.

  Return type: SETOF record

Name	Type	Description
file	text	Name of the file where the memory is applied for.
line	int8	Line number of the code in the file where the requested memory is located.
size	int8	Size of the allocated memory, in bytes. The value is accumulated if the memory is allocated for multiple times to the same line of the same file.

NOTE

This view takes effect only in thread pool mode and is not supported in the Lite release version.

- gs_get_history_memory_detail(cstring)

  Description: Queries historical memory snapshot information. The input parameter type is cstring. The value can be **NULL** or the name of the memory snapshot log file.

  a. If the value of the input parameter is **NULL**, the list of all memory snapshot log files on the current node is displayed.
  b. If the value of the input parameter is the name of the memory snapshot log file in the list queried in **a**, the detailed information about the memory snapshot recorded in the log file is displayed.
  c. If you enter any other input parameter, the system displays a message indicating that the input parameter is incorrect or the file fails to be opened.

  To query this function, you must have the sysadmin or monitor admin permission.

  Return type: text

Name	Type	Description
memory_info	text	Memory information. If the input parameter of the function is set to **NULL**, the memory snapshot file list is displayed. If the input parameter is set to the name of the memory snapshot file, the content of the file is displayed.

- gs_stack()

  Description: Displays the call stack of a thread. To query this function, you must have the sysadmin or monadmin permission.

  Parameter: tid, which indicates the thread ID. **tid** is an optional parameter. If it is specified, the function returns the call stack of the thread corresponding to **tid**. If it is not specified, the function returns the call stacks of all threads.

  Return value: If **tid** is specified, the return value is of the TEXT type. If **tid** is not specified, the return value is a SETOF record.

  Example:

```
openGauss=# SELECT gs_stack(139663481165568);
                          gs_stack
--------------------------------------------------------------------
 __poll + 0x2d                                              +
 WaitLatchOrSocket(Latch volatile*, int, int, long) + 0x29f        +
 WaitLatch(Latch volatile*, int, long) + 0x2e                      +
 JobScheduleMain() + 0x90f                                         +
 int GaussDbThreadMain<(knl_thread_role)9>(knl_thread_arg*) + 0x456+
 InternalThreadFunc(void*) + 0x2d                                  +
 ThreadStarterFunc(void*) + 0xa4                                   +
 start_thread + 0xc5                                               +
 clone + 0x6d                                                      +
(1 row)
```

- gs_get_thread_memctx_detail(tid,text)

  Description: Returns the memory application details of the specified memory context, including the file, line number, and size of each memory application (the size of the same line in the same file is accumulated). Only the memory context queried through the **gs_thread_memory_context** view is supported.

The first input parameter is the thread ID (the **tid** column of the data returned by the **gs_thread_memory_context**), and the second parameter is the memory context name (the **contextname** column of the data returned by **gs_thread_memory_context**). To query this function, you must have the sysadmin or monitor admin permission.

Return type: SETOF record

Name	Type	Description
file	text	Name of the file where the memory is applied for.
line	int8	Line number of the code in the file where the requested memory is located.
size	int8	Size of the allocated memory, in bytes. The value is accumulated if the memory is allocated for multiple times to the same line of the same file.

NOTE

This view is not supported in the Lite release version.

11.5.28 Trigger Functions

- pg_get_triggerdef(oid)

  Description: Obtains the definition information of a trigger.

  Parameter: OID of the trigger to be queried

  Return type: text

  Example:

```
openGauss=# select pg_get_triggerdef(oid) from pg_trigger;
                                                                   pg_get_triggerdef
-------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------
 CREATE TRIGGER tg1 BEFORE INSERT ON gtest26 FOR EACH STATEMENT EXECUTE PROCEDURE
gtest_trigger_func()
 CREATE TRIGGER tg03 AFTER INSERT ON gtest26 FOR EACH ROW WHEN ((new.a IS NOT NULL))
EXECUTE PROCEDURE gtest_trigger_func()
(2 rows)
```

- pg_get_triggerdef(oid, boolean)

  Description: Obtains the definition information of a trigger.

  Parameter: OID of the trigger to be queried and whether it is displayed in pretty mode

  NOTE

  Boolean parameters take effect only when the WHEN condition is specified during trigger creation.

  Return type: text

  Example:

```
openGauss=# select pg_get_triggerdef(oid,true) from pg_trigger;
                                                                   pg_get_triggerdef
```

```
------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------
 CREATE TRIGGER tg1 BEFORE INSERT ON gtest26 FOR EACH STATEMENT EXECUTE PROCEDURE
gtest_trigger_func()
 CREATE TRIGGER tg03 AFTER INSERT ON gtest26 FOR EACH ROW WHEN (new.a IS NOT NULL)
EXECUTE PROCEDURE gtest_trigger_func()
(2 rows)

openGauss=# select pg_get_triggerdef(oid,false) from pg_trigger;
                                                                    pg_get_triggerdef
------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------
 CREATE TRIGGER tg1 BEFORE INSERT ON gtest26 FOR EACH STATEMENT EXECUTE PROCEDURE
gtest_trigger_func()
 CREATE TRIGGER tg03 AFTER INSERT ON gtest26 FOR EACH ROW WHEN ((new.a IS NOT NULL))
EXECUTE PROCEDURE gtest_trigger_func()
(2 rows)
```

11.5.29 Hash Function

- hash_array(anyarray)

  Description: Hashes an array, obtains the result of an array element using the hash function, and returns the combination result.

  Parameter: data of the anyarray type

  Return type: integer

  Example:

```
openGauss=# select hash_array(ARRAY[[1,2,3],[1,2,3]]);
 hash_array
------------
 -382888479
(1 row)
```

- hash_group(key)

  Description: Calculates the hash value of each column in the Group Clause in the streaming engine. (Due to specification changes, the current version no longer supports the current feature. Do not use this feature.)

  Parameter: **key** indicates the value of each column in the Group Clause.

  Return type: 32-bit hash value

  Example:

```
Perform the following steps in sequence.
openGauss=# CREATE TABLE tt(a int, b int,c int,d int);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
openGauss=# select * from tt;
 a | b | c | d
---+---+---+---
(0 rows)

openGauss=# insert into tt values(1,2,3,4);
INSERT 0 1
openGauss=# select * from tt;
 a | b | c | d
---+---+---+---
 1 | 2 | 3 | 4
(1 row)

openGauss=# insert into tt values(5,6,7,8);
INSERT 0 1
openGauss=# select * from tt;
 a | b | c | d
```

```
---+---+---+---
 1 | 2 | 3 | 4
 5 | 6 | 7 | 8
(2 rows)

openGauss=# select hash_group(a,b) from tt where a=1 and b=2;
 hash_group
------------
  990882385
(1 row)
```

- hash_numeric(numeric)

  Description: Calculates the hash value of numeric data.

  Parameter: data of the numeric type.

  Return type: integer

  Example:

```
openGauss=# select hash_numeric(30);
 hash_numeric
--------------
   -282860963
(1 row)
```

- hash_range(anyrange)

  Description: Calculates the hash value of a range.

  Parameter: data of the anyrange type

  Return type: integer

  Example:

```
openGauss=# select hash_range(numrange(1.1,2.2));
 hash_range
------------
  683508754
(1 row)
```

- hashbpchar(character)

  Description: Calculates the hash value of bpchar.

  Parameter: data of the character type

  Return type: integer

  Example:

```
openGauss=# select hashbpchar('hello');
 hashbpchar
-------------
 -1870292951
(1 row)
```

- hashchar(char)

  Description: Converts char and Boolean data into hash values.

  Parameter: data of the char or bool type

  Return type: integer

  Example:

```
openGauss=# select hashbpchar('hello');
 hashbpchar
-------------
 -1870292951
(1 row)

openGauss=# select hashchar('true');
  hashchar
------------
```

```
 1686226652
(1 row)
```

- hashenum(anyenum)

  Description: Converts enumerated values to hash values.

  Parameter: data of the anyenum type

  Return type: integer

  Example:

```
openGauss=# CREATE TYPE b1 AS ENUM('good', 'bad', 'ugly');
CREATE TYPE
openGauss=# call hashenum('good'::b1);
  hashenum
------------
 1821213359
(1 row)
```

- hashfloat4(real)

  Description: Converts float4 values to hash values.

  Parameter: data of the real type

  Return type: integer

  Example:

```
openGauss=# select hashfloat4(12.1234);
 hashfloat4
------------
 1398514061
(1 row)
```

- hashfloat8(double precision)

  Description: Converts float8 values to hash values.

  Parameter: data of the double precision type

  Return type: integer

  Example:

```
openGauss=# select hashfloat8(123456.1234);
 hashfloat8
------------
 1673665593
(1 row)
```

- hashinet(inet)

  Description: Supports hashing indexes on inet or cidr. Returns the hash value of inet.

  Parameter: data of the inet type

  Return type: integer

  Example:

```
openGauss=# select hashinet('127.0.0.1'::inet);
  hashinet
-------------
 -1435793109
(1 row)
```

- hashint1(tinyint)

  Description: Converts INT1 values to hash values.

  Parameter: data of the tinyint type

  Return type: uint32

  Example:

```
openGauss=# select hashint1(20);
  hashint1
-------------
 -2014641093
(1 row)
```

- hashint2(smallint)

  Description: Converts INT2 values to hash values.

  Parameter: data of the smallint type

  Return type: uint32

  Example:

```
openGauss=# select hashint2(20000);
  hashint2
------------
 -863179081
(1 row)
```

11.5.30 Prompt Message Function

- report_application_error

  Description: This function can be used to throw errors during PL execution.

  Return type: void

Table 11-87 report_application_error parameter description

Parameter	Type	Description	Mandatory or Not
log	text	Content of an error message.	Yes
code	int4	Error code corresponding to an error message. The value ranges from –20999 to –20000.	No

11.5.31 Global Temporary Table Functions

- pg_get_gtt_relstats(relOid)

  Description: Displays basic information about a global temporary table specified by the current session.

  Parameter: OID of the global temporary table

  Return type: record

  Example:

```
openGauss=# select * from pg_get_gtt_relstats(74069);
 relfilenode | relpages | reltuples | relallvisible | relfrozenxid | relminmxid
-------------+----------+-----------+---------------+--------------+------------
       74069 |       58 |     13000 |             0 |        11151 |          0
(1 row)
```

- pg_get_gtt_statistics(relOid, attnum, ''::text)

  Description: Displays statistics about a single column in a global temporary table specified by the current session.

  Parameter: OID and the **attnum** attribute of the global temporary table

  Return type: record

Example:

```
openGauss=# select * from pg_get_gtt_statistics(74069,1,''::text);
 starelid | starelkind | staattnum | stainherit | stanullfrac | stawidth | stadistinct | stakind1 | stakind2 |
stakind3 | stakind4 | stakind5 | staop1 | staop2 | staop3 | staop4 | staop5 | stanumbers1 | stanumbers2
| stanumbers3 | stanu
mbers4 | stanumbers5
|


stavalues1

                                                                  | stavalues2 | stavalues3 | stavalues4 | stavalues5 |
stadndistinct | staextinfo
----------+------------+-----------+------------+-------------+----------+-------------+----------+----------
+----------+----------+----------+--------+--------+--------+--------+--------+-------------+-------------
+-------------+------
-------+-------------
+------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------+------------+------------+------------
+------------+---------------+------------
    74069 | c          |         1 | f          |           0 |        4 |          -1 |        2 |        3 |        0 |        0 |
0 |     97 |     97 |      0 |      0 |      0 |             | {1}         |             |
       |             |
{1,130,260,390,520,650,780,910,1040,1170,1300,1430,1560,1690,1820,1950,2080,2210,2340,2470,2600,
2730,2860,2990,3120,3250,3380,3510,3640,3770,3900,4030,4160,4290,4420,4550,4680,4810,4940,5070
,5200,5330,5460,5590,57
20,5850,5980,6110,6240,6370,6500,6630,6760,6890,7020,7150,7280,7410,7540,7670,7800,7930,8060,8
190,8320,8450,8580,8710,8840,8970,9100,9230,9360,9490,9620,9750,9880,10010,10140,10270,10400,1
0530,10660,10790,10920,11050,11180,11310,11440,1
1570,11700,11830,11960,12090,12220,12350,12480,12610,12740,12870,13000} |            |
|            |            |             0 |
(1 row)
```

- pg_gtt_attached_pid(relOid)

  Description: Displays PIDs and session IDs of all threads that are using the specified global temporary table. If the thread pool is enabled and the session that is using the global temporary table is in the detach state, then the PID is **0** and **sessionid** indicates the session ID.

  Parameter: OID of the global temporary table

  Return type: record

  Example:

```
openGauss=# select * from pg_gtt_attached_pid(74069);
 relid |       pid       |   sessionid
-------+-----------------+-----------------
 74069 | 139648170456832 | 139648170456832
 74069 | 139648123270912 | 139648123270912
(2 rows)
```

- dbe_perf.get_global_full_sql_by_timestamp(start_timestamp timestamp, end_timestamp timestamp)

  Description: Obtains full SQL information at the instance level.

  Return type: record

Table 11-88 dbe_perf.get_global_full_sql_by_timestamp parameter description

Parameter	Type	Description
start_timestamp	timestamp	Start point of the SQL start time range.
end_timestamp	timestamp	End point of the SQL start time range.

- dbe_perf.get_global_slow_sql_by_timestamp(start_timestamp timestamp, end_timestamp timestamp)

  Description: Obtains slow SQL information at the instance level.

  Return type: record

Table 11-89 dbe_perf.get_global_slow_sql_by_timestamp parameter description

Parameter	Type	Description
start_timestamp	timestamp	Start point of the SQL start time range.
end_timestamp	timestamp	End point of the SQL start time range.

- statement_detail_decode(detail text, format text, pretty bool)

  Parses the details column in a full or slow SQL statement.

Table 11-90 statement_detail_decode parameter description

Parameter	Type	Description
detail	text	Set of events generated by the SQL statement (unreadable).
format	text	Parsing output format. The value is **plaintext** or **json**.
pretty	bool	Whether to display the text in pretty format when **format** is set to **plaintext**. The options are as follows: ● The value **true** indicates that **\n** is used to separate events. ● The value **false** indicates that events are separated by commas (,).

- pg_list_gtt_relfrozenxids()

  Description: Displays the frozen XID of each session.

  If the value of **pid** is **0**, the earliest frozen XID of all sessions is displayed.

  Parameter: none

Return type: record

Example:

```
openGauss=# select * from pg_list_gtt_relfrozenxids();
      pid       | relfrozenxid
-----------------+--------------
 139648123270912 |        11151
 139648170456832 |        11155
               0 |        11151
(3 rows)
```

11.5.32 Fault Injection System Function

- gs_fault_inject(int64, text, text, text, text, text)

  Description: This function cannot be called. WARNING information "unsupported fault injection" is reported when this function is called, which does not affect or change the database.

  Parameter: fault injection of the int64 type (**0**: CLOG extended page; **1**: CLOG page reading; **2**: forcible deadlock)

  – If the first input parameter of text is set to **2** and the second input parameter of text is set to **1**, the second input parameter deadlock occurs. Other input parameters are not deadlocked. When the first input parameter is **0** or **1**, the second input parameter indicates the number of the start page from which the CLOG starts to be extended or read.

  – The third input parameter of text indicates the number of extended or read pages when the first input parameter is **0** or **1**.

  – The fourth to sixth input parameters of text are reserved.

  Return type: int64

11.5.33 Dynamic Data Masking Functions

NOTE

This function is an internal function. For details, see "Database Security > Dynamic Data Anonymization" in *Feature Description*.

- creditcardmasking(col text, letter char default 'x')

  Description: Replaces the digits before the last four bits following the col string with letters.

  Parameter: Character string to be replaced or character string used for replacement

  Return type: text

- basicmailmasking(col text, letter char default 'x')

  Description: Replaces the characters before the first at sign (@) in the col string with letters.

  Parameter: Character string to be replaced or character string used for replacement

  Return type: text

- fullmailmasking(col text, letter char default 'x')

  Description: Replaces the characters (except **@**) before the last period (.) in the col string with letters.

Parameter: Character string to be replaced or character string used for replacement

Return type: text

- alldigitsmasking(col text, letter char default '0')

  Description: Replaces the digits in the col string with letters.

  Parameter: Character string to be replaced or character string used for replacement

  Return type: text

- shufflemasking(col text)

  Description: Sorts the characters in the col string out of order.

  Parameter: Character string to be replaced or character string used for replacement

  Return type: text

- randommasking(col text)

  Description: Randomizes the characters in the col string.

  Parameter: Character string to be replaced or character string used for replacement

  Return type: text

- regexpmasking

  Description: Specifies the internal function of the masking policy, which is used to replace characters using a regular expression.

  Parameter: col text, reg text, replace_text text, pos INTEGER default 0, reg_len INTEGER default -1

  Return type: text

11.5.34 Hierarchical Recursion Query Functions

The following functions can be used in a hierarchical recursion query statement to return information about the connection path.

- sys_connect_by_path(col, separator)

  Description: Returns the connection path from the root node to the current row. This function applies only to hierarchical recursion queries.

  The **col** parameter indicates the name of the column displayed in the path. Only columns of the CHAR, VARCHAR, NVARCHAR2, and TEXT types are supported. The **separator** parameter indicates the separator between path nodes.

  Return type: text

  Example:

```
openGauss=# select *, sys_connect_by_path(name, '-') from connect_table start with id = 1 connect by prior id = pid;
 id | pid | name | sys_connect_by_path
----+-----+------+---------------------
  1 |   0 | a    | -a
  2 |   1 | b    | -a-b
  4 |   1 | d    | -a-d
  3 |   2 | c    | -a-b-c
(4 rows)
```

- connect_by_root(col)

  Description: Returns the value of a column in the top-most parent row of the current row. This function applies only to hierarchical recursion queries.

  The **col** parameter indicates the name of an output column.

  Return type: data type of the specified column **col**.

  Example:

```
openGauss=# select *, connect_by_root(name) from connect_table start with id = 1 connect by prior
id = pid;
 id | pid | name | connect_by_root
----+-----+------+-----------------
  1 |   0 | a    | a
  2 |   1 | b    | a
  4 |   1 | d    | a
  3 |   2 | c    | a
(4 rows)
```

11.5.35 Other System Functions

The built-in functions and operators of GaussDB are inherited from the open-source PostgreSQL. For details about the following functions, see the **office PostgreSQL documents**.

_pg_char_max_length	_pg_char_octet_length	_pg_datetime_precision	_pg_expandarray	_pg_index_position	_pg_interval_type	_pg_numeric_precision
_pg_numeric_precision_radix	_pg_numeric_scale	_pg_truetypid	_pg_truetypmod	abbrev	abs	abstime
abstimeeq	abstimege	abstimegt	abstimein	abstimele	abstimelt	abstimene
abstimeout	abstimerecv	abstimesend	aclcontains	acldefault	aclexplode	aclinsert
aclitemeq	aclitemin	aclitemout	aclremove	acos	age	akeys
any_in	any_out	anyarray_in	anyarray_out	anyarray_recv	anyarray_send	anyelement_in
anyelement_out	anyenum_in	anyenum_out	anynonarray_in	anynonarray_out	anyrange_in	anyrange_out
anytextcat	area	areajoinsel	areasel	array_agg	array_agg_finalfn	array_agg_transfn
array_append	array_cat	array_dims	array_eq	array_fill	array_ge	array_gt
array_in	array_larger	array_le	array_length	array_lower	array_lt	array_ndims

array_ne	array_out	array_prepend	array_recv	array_send	array_smaller	array_to_json
array_to_string	array_typanalyze	array_upper	arraycontained	arraycontains	arraycontjoinsel	arraycontsel
arrayoverlap	ascii	asin	atan	atan2	avals	avg
big5_to_euc_tw	big5_to_mic	big5_to_utf8	bit	bit_and	bit_in	bit_length
bit_or	bit_out	bit_recv	bit_send	bitand	bitcat	bitcmp
biteq	bitge	bitgt	bitle	bitlt	bitne	bitnot
bitor	bitshiftleft	bitshiftright	bittypmodin	bittypmodout	bitxor	bool
bool_and	bool_or	booland_statefunc	booleq	boolge	boolgt	boolin
boolle	boollt	boolne	boolor_statefunc	boolout	boolrecv	boolsend
box	box_above	box_above_eq	box_add	box_below	box_below_eq	box_center
box_contain	box_contain_pt	box_contained	box_distance	box_div	box_eq	box_ge
box_gt	box_in	box_intersect	box_le	box_left	box_lt	box_mul
box_out	box_overabove	box_overbelow	box_overlap	box_overleft	box_overright	box_recv
box_right	box_same	box_send	box_sub	bpchar	bpchar_larger	bpchar_pattern_ge
bpchar_pattern_gt	bpchar_pattern_le	bpchar_pattern_lt	bpchar_smaller	bpchar_sortsupport	bpcharcmp	bpchareq
bpcharge	bpchargt	bpchariclike	bpcharicnlike	bpcharicregexeq	bpcharicregexne	bpcharin
bpcharle	bpcharlike	bpcharlt	bpcharne	bpcharnlike	bpcharout	bpcharrecv
bpcharregexeq	bpcharregexne	bpcharsend	bpchartypmodin	bpchartypmodout	broadcast	btabstimecmp

btarraycmp	btbeginscan	btboolcmp	btbpchar_pattern_cmp	btbuild	btbuildempty	btbulkdelete
btcanreturn	btcharcmp	btcostestimate	btendscan	btfloat48cmp	btfloat4cmp	btfloat4sortsupport
btfloat84cmp	btfloat8cmp	btfloat8sortsupport	btgetbitmap	btgettuple	btinsert	btint24cmp
btint28cmp	btint2cmp	btint2sortsupport	btint42cmp	btint48cmp	btint4cmp	btint4sortsupport
btint82cmp	btint84cmp	btint8cmp	btint8sortsupport	btmarkpos	btnamecmp	btnamesortsupport
btoidcmp	btoidsortsupport	btoidvectorcmp	btoptions	btrecordcmp	btreltimecmp	btrescan
btrestrpos	btrim	bttext_pattern_cmp	bttextcmp	bttextsortsupport	bttidcmp	bttintervalcmp
btvacuumcleanup	bytea_sortsupport	bytea_string_agg_finalfn	bytea_string_agg_transfn	byteacat	byteacmp	byteaeq
byteage	byteagt	byteain	byteale	bytealike	bytealt	byteane
byteanlike	byteaout	bytearecv	byteasend	cash_cmp	cash_div_cash	cash_div_flt4
cash_div_flt8	cash_div_int2	cash_div_int4	cash_div_int8	cash_eq	cash_ge	cash_gt
cash_in	cash_le	cash_lt	cash_mi	cash_mul_flt4	cash_mul_flt8	cash_mul_int2
cash_mul_int4	cash_mul_int8	cash_ne	cash_out	cash_pl	cash_recv	cash_send
cashlarger	cashsmaller	cbrt	ceil	ceiling	center	char
char_length	character_length	chareq	charge	chargt	charin	charle
charlt	charne	charout	charrecv	charsend	chr	cideq

cidin	cidout	cidr	cidr_in	cidr_out	cidr_recv	cidr_send
cidrecv	cidsend	circle	circle_above	circle_add_pt	circle_below	circle_center
circle_contain	circle_contain_pt	circle_contained	circle_distance	circle_div_pt	circle_eq	circle_ge
circle_gt	circle_in	circle_le	circle_left	circle_lt	circle_mul_pt	circle_ne
circle_out	circle_overabove	circle_overbelow	circle_overlap	circle_overleft	circle_overright	circle_recv
circle_right	circle_same	circle_send	circle_sub_pt	clock_timestamp	close_lb	close_ls
close_lseg	close_pb	close_pl	close_ps	close_sb	close_sl	col_description
concat	concat_ws	contjoinsel	contsel	convert	convert_from	convert_to
corr	cos	cot	count	covar_pop	covar_samp	cstring_in
cstring_out	cstring_recv	cstring_send	cume_dist	current_database	current_query	current_schema
xpath_exists	current_setting	current_user	currtid	currtid2	currval	cursor_to_xml
cursor_to_xmlschema	database_to_xml	database_to_xml_and_xmlschema	database_to_xmlschema	date	date_cmp	date_cmp_timestamp
date_cmp_timestamptz	date_eq	date_eq_timestamp	date_eq_timestamptz	date_ge	date_ge_timestamp	date_ge_timestamptz
date_gt	date_gt_timestamp	date_gt_timestamptz	date_in	date_larger	date_le	date_le_timestamp
date_le_timestamptz	date_lt	date_lt_timestamp	date_lt_timestamptz	date_mi	date_mi_interval	date_mii
date_ne	date_ne_timestamp	date_ne_timestamptz	date_out	date_pl_interval	date_pli	date_recv

date_send	date_smaller	date_sortsupport	daterange_canonical	daterange_subdiff	datetime_pl	datetimetz_pl
dcbrt	decode	defined	degrees	delete	dense_rank	dexp
diagonal	diameter	dispell_init	dispell_lexize	dist_cpoly	dist_lb	dist_pb
dist_pc	dist_pl	dist_ppath	dist_ps	dist_sb	dist_sl	div
dlog1	dlog10	domain_in	domain_recv	dpow	dround	dsimple_init
dsimple_lexize	dsnowball_init	dsnowball_lexize	dsqrt	dsynonym_init	dsynonym_lexize	dtrunc
each	enum_ne	enum_out	enum_range	enum_recv	enum_send	enum_smaller
eqjoinsel	eqsel	euc_cn_to_mic	euc_cn_to_utf8	euc_jis_2004_to_shift_jis_2004	euc_jis_2004_to_utf8	euc_jp_to_mic
euc_jp_to_sjis	euc_jp_to_utf8	euc_kr_to_mic	euc_kr_to_utf8	euc_tw_to_big5	euc_tw_to_mic	euc_tw_to_utf8
every	exist	exists_all	exists_any	exp	factorial	family
fdw_handler_in	fdw_handler_out	fetchval	first_value	float4	float4_accum	float48div
float48eq	float48ge	float48gt	float48le	float48lt	float48mi	float48mul
float48ne	float48pl	float4abs	float4div	float4eq	float4ge	float4gt
float4in	float4larger	float4le	float4lt	float4mi	float4mul	float4ne
float4out	float4pl	float4recv	float4send	float4smaller	float4um	float4up
float8	float8_accum	float8_avg	float8_collect	float8_corr	float8_covar_pop	float8_covar_samp
float8_regr_accum	float8_regr_avgx	float8_regr_avgy	float8_regr_collect	float8_regr_intercept	float8_regr_r2	float8_regr_slope

float8_regr_sxx	float8_regr_sxy	float8_regr_syy	float8_stddev_pop	float8_stddev_samp	float8_var_pop	float8_var_samp
float84div	float84eq	float84ge	float84gt	float84le	float84lt	float84mi
float84mul	float84ne	float84pl	float8abs	float8div	float8eq	float8ge
float8gt	float8in	float8larger	float8le	float8lt	float8mi	float8mul
float8ne	float8out	float8pl	float8recv	float8send	float8smaller	float8um
float8up	floor	flt4_mul_cash	flt8_mul_cash	fmgr_c_validator	fmgr_internal_validator	fmgr_sql_validator
format	format_type	gb18030_to_utf8	gbk_to_utf8	generate_series	generate_subscripts	get_bit
get_byte	get_current_ts_config	-	-	gin_clean_pending_list	gin_cmp_prefix	gin_cmp_tslexeme
gin_extract_tsquery	gin_extract_tsvector	gin_tsquery_consistent	gin_tsquery_triconsistent	ginarrayconsistent	ginarrayextract	ginarraytriconsistent
ginbeginscan	ginbuild	ginbuildempty	ginbulkdelete	gincostestimate	ginendscan	gingetbitmap
gininsert	ginmarkpos	ginoptions	ginqueryarrayextract	ginrescan	ginrestrpos	ginvacuumcleanup
gist_box_compress	gist_box_consistent	gist_box_decompress	gist_box_penalty	gist_box_picksplit	gist_box_same	gist_box_union
gist_circle_compress	gist_circle_consistent	gist_point_compress	gist_point_consistent	gist_point_distance	gist_poly_compress	gist_poly_consistent
gistbeginscan	gistbuild	gistbuildempty	gistbulkdelete	gistcostestimate	gistendscan	gistgetbitmap
gistgettuple	gistinsert	gistmarkpos	gistoptions	gistrescan	gistrestrpos	gistvacuumcleanup

gtsquery_compress	gtsquery_consistent	gtsquery_decompress	gtsquery_penalty	gtsquery_picksplit	gtsquery_same	gtsquery_union
gtsvector_compress	gtsvector_consistent	gtsvector_decompress	gtsvector_penalty	gtsvector_picksplit	gtsvector_same	gtsvector_union
gtsvectorin	gtsvectorout	has_tablespace_privilege	has_type_privilege	hash_aclitem	hashbeginscan	hashbuild
hashbuildempty	hashbulkdelete	hashcostestimate	hashendscan	hashgetbitmap	hashgettuple	hashinsert
hashint2vector	hashint4	hashint8	hashmacaddr	hashmarkpos	hashname	hashoid
hashoidvector	hashoptions	hashrescan	hashrestrpos	hashtext	hashvacuumcleanup	hashvarlena
host	hostmask	iclikejoinsel	iclikesel	icnlikejoinsel	icnlikesel	icregexeqjoinsel
icregexeqsel	icregexnejoinsel	icregexnesel	inet_client_addr	inet_client_port	inet_in	inet_out
inet_recv	inet_send	inet_server_addr	inet_server_port	inetand	inetmi	inetmi_int8
inetnot	inetor	inetpl	initcap	int2_accum	int2_avg_accum	int2_mul_cash
int2_sum	int24div	int24eq	int24ge	int24gt	int24le	int24lt
int24mi	int24mul	int24ne	int24pl	int28div	int28eq	int28ge
int28gt	int28le	int28lt	int28mi	int28mul	int28ne	int28pl
int2abs	int2and	int2div	int2eq	int2ge	int2gt	int2in
int2larger	int2le	int2lt	int2mi	int2mod	int2mul	int2ne
int2not	int2or	int2out	int2pl	int2recv	int2send	int2shl
int2shr	int2smaller	int2um	int2up	int2vectoreq	int2vectorin	int2vectorout

int2vectorrecv	int2vectorsend	int2xor	int4_accum	int4_avg_accum	int4_mul_cash	int4_sum
int42div	int42eq	int42ge	int42gt	int42le	int42lt	int42mi
int42mul	int42ne	int42pl	int48div	int48eq	int48ge	int48gt
int48le	int48lt	int48mi	int48mul	int48ne	int48pl	int4abs
int4and	int4div	int4eq	int4ge	int4gt	int4in	int4inc
int4larger	int4le	int4lt	int4mi	int4mod	int4mul	int4ne
int4not	int4or	int4out	int4pl	int4range	int4range_canonical	int4range_subdiff
int4recv	int4send	int4shl	int4shr	int4smaller	int4um	int4up
int4xor	int8	int8_avg	int8_avg_accum	int8_avg_collect	int8_mul_cash	int8_sum
int8_sum_to_int8	int8_accum	int82div	int82eq	int82ge	int82gt	int82le
int82lt	int82mi	int82mul	int82ne	int82pl	int84div	int84eq
int84ge	int84gt	int84le	int84lt	int84mi	int84mul	int84ne
int84pl	int8abs	int8and	int8div	int8eq	int8ge	int8gt
int8in	int8inc	int8inc_any	int8inc_float8_float8	int8larger	int8le	int8lt
int8mi	int8mod	int8mul	int8ne	int8not	int8or	int8out
int8pl	int8pl_inet	int8range	int8range_canonical	int8range_subdiff	int8recv	int8send
int8shl	int8shr	int8smaller	int8um	int8up	int8xor	integer_pl_date
inter_lb	inter_sb	inter_sl	internal_in	internal_out	interval	interval_accum
interval_avg	interval_cmp	interval_collect	interval_div	interval_eq	interval_ge	interval_gt

interval_hash	interval_in	interval_larger	interval_le	interval_lt	interval_mi	interval_mul
interval_ne	interval_out	interval_pl	interval_pl_date	interval_pl_time	interval_pl_timestamp	interval_pl_timestamptz
interval_pl_timetz	interval_recv	interval_send	interval_smaller	interval_transform	interval_um	intervaltypmodin
intervaltypmodout	intinterval	isexists	ishorizontal	iso_to_koi8r	iso_to_mic	iso_to_win1251
iso_to_win866	iso8859_1_to_utf8	iso8859_to_utf8	isparallel	isperp	isvertical	johab_to_utf8
jsonb_in	jsonb_out	jsonb_recv	jsonb_send	-	-	-
json_in	json_out	json_recv	json_send	justify_days	justify_hours	justify_interval
koi8r_to_iso	koi8r_to_mic	koi8r_to_utf8	koi8r_to_win1251	koi8r_to_win866	koi8u_to_utf8	language_handler_in
language_handler_out	latin1_to_mic	latin2_to_mic	latin2_to_win1250	latin3_to_mic	latin4_to_mic	like_escape
likejoinsel	likesel	line	line_distance	line_eq	line_horizontal	line_in
line_interpt	line_intersect	line_out	line_parallel	line_perp	line_recv	line_send
line_vertical	ln	lo_close	lo_creat	lo_create	lo_export	lo_import
lo_lseek	lo_open	lo_tell	lo_truncate	lo_unlink	log	loread
lower	lower_inc	lower_inf	lowrite	lpad	lseg	lseg_center
lseg_distance	lseg_eq	lseg_ge	lseg_gt	lseg_horizontal	lseg_in	lseg_interpt
lseg_intersect	lseg_le	lseg_length	lseg_lt	lseg_ne	lseg_out	lseg_parallel
lseg_perp	lseg_recv	lseg_send	lseg_vertical	ltrim	macaddr_and	macaddr_cmp

macaddr_eq	macaddr_ge	macaddr_gt	macaddr_in	macaddr_le	macaddr_lt	macaddr_ne
macaddr_not	macaddr_or	macaddr_out	macaddr_recv	macaddr_send	makeaclitem	masklen
max	md5 (The MD5 encryption algorithm has lower security and poses security risks. Therefore, you are advised to use a more secure encryption algorithm.)	mic_to_big5	mic_to_euc_cn	mic_to_euc_jp	mic_to_euc_kr	mic_to_euc_tw
mic_to_iso	mic_to_koi8r	mic_to_latin1	mic_to_latin2	mic_to_latin3	mic_to_latin4	mic_to_sjis
mic_to_win1250	mic_to_win1251	mic_to_win866	min	mktinterval	money	mul_d_interval
name	nameeq	namege	namegt	nameiclike	nameicnlike	nameicregexeq
nameicregexne	namein	namele	namelike	namelt	namene	namenlike
nameout	namerecv	nameregexeq	nameregexne	namesend	neqjoinsel	neqsel
network_cmp	network_eq	network_ge	network_gt	network_le	network_lt	network_ne
network_sub	network_subeq	network_sup	network_supeq	nlikejoinsel	nlikesel	numeric

numeric_abs	numeric_accum	numeric_add	numeric_avg	numeric_avg_accum	numeric_avg_collect	numeric_cmp
numeric_collect	numeric_div	numeric_div_trunc	numeric_eq	numeric_exp	numeric_fac	numeric_ge
numeric_gt	numeric_in	numeric_inc	numeric_larger	numeric_le	numeric_ln	numeric_log
numeric_lt	numeric_mod	numeric_mul	numeric_ne	numeric_out	numeric_power	numeric_recv
numeric_send	numeric_smaller	numeric_sortsupport	numeric_sqrt	numeric_stddev_pop	numeric_stddev_samp	numeric_sub
numeric_transform	numeric_uminus	numeric_uplus	numeric_var_pop	numeric_var_samp	numerictypmodin	numerictypmodout
numrange_subdiff	oid	oideq	oidge	oidgt	oidin	oidlarger
oidle	oidlt	oidne	oidout	oidrecv	oidsend	oidsmaller
oidvectoreq	oidvectorge	oidvectorgt	oidvectorin	oidvectorle	oidvectorlt	oidvectorne
oidvectorout	oidvectorrecv	oidvectorsend	oidvectortypes	on_pb	on_pl	on_ppath
on_ps	on_sb	on_sl	opaque_in	opaque_out	ordered_set_transition	overlaps
overlay	path	path_add	path_add_pt	path_center	path_contain_pt	path_distance
path_div_pt	path_in	path_inter	path_length	path_mul_pt	path_n_eq	path_n_ge
path_n_gt	path_n_le	path_n_lt	path_npoints	path_out	path_recv	path_send
path_sub_pt	percentile_cont	percentile_cont_float8_final	percentile_cont_interval_final	pg_char_to_encoding	pg_cursor	pg_encoding_max_length

pg_encoding_to_char	pg_extension_config_dump	-	-	pg_node_tree_in	pg_node_tree_out	pg_node_tree_recv
pg_node_tree_send	pg_prepared_statement	pg_prepared_xact	-	-	pg_show_all_settings	pg_stat_get_bgwriter_stat_reset_time
pg_stat_get_buf_fsync_backend	pg_stat_get_checkpoint_sync_time	pg_stat_get_checkpoint_write_time	pg_stat_get_db_blk_read_time	pg_stat_get_db_blk_write_time	pg_stat_get_db_conflict_all	pg_stat_get_db_conflict_bufferpin
pg_stat_get_db_conflict_snapshot	pg_stat_get_db_conflict_startup_deadlock	pg_switch_xlog	xpath	pg_timezone_abbrevs	pg_timezone_names	pg_stat_get_wal_receiver
plpgsql_call_handler	plpgsql_inline_handler	plpgsql_validator	point_above	point_add	point_below	point_distance
point_div	point_eq	point_horiz	point_in	point_left	point_mul	point_ne
point_out	point_recv	point_right	point_send	point_sub	point_vert	poly_above
poly_below	poly_center	poly_contain	poly_contain_pt	poly_contained	poly_distance	poly_in
poly_left	poly_npoints	poly_out	poly_overabove	poly_overbelow	poly_overlap	poly_overleft
poly_overright	poly_recv	poly_right	poly_same	poly_send	polygon	position
positionjoinsel	positionsel	postgresql_fdw_validator	pow	power	prsd_end	prsd_headline
prsd_lextype	prsd_nexttoken	prsd_start	pt_contained_circle	pt_contained_poly	query_to_xml	query_to_xml_and_xmlschema
query_to_xmlschema	quote_ident	quote_literal	quote_nullable	radians	radius	random

range_adjacent	range_after	range_before	range_cmp	range_contained_by	range_contains	range_contains_elem
range_eq	range_ge	range_gist_compress	range_gist_consistent	range_gist_decompress	range_gist_penalty	range_gist_picksplit
range_gist_same	range_gist_union	range_gt	range_in	range_intersect	range_le	range_lt
range_minus	range_ne	range_out	range_overlaps	range_overleft	range_overright	range_recv
range_send	range_typanalyze	range_union	rank	record_eq	record_ge	record_gt
record_in	record_le	record_lt	record_ne	record_out	record_recv	record_send
regclass	regclassin	regclassout	regclassrecv	regclasssend	regconfigin	regconfigout
regconfigrecv	regconfigsend	regdictionaryin	regdictionaryout	regdictionaryrecv	regdictionarysend	regexeqjoinsel
regexeqsel	regexnejoinsel	regexnesel	regexp_matches	regexp_replace	regexp_split_to_array	regexp_split_to_table
regoperatorin	regoperatorout	regoperatorrecv	regoperatorsend	regoperin	regoperout	regoperrecv
regopersend	regprocedurein	regprocedureout	regprocedurerecv	regproceduresend	regprocin	regprocout
regprocrecv	regprocsend	regr_avgx	regr_avgy	regr_count	regr_intercept	regr_r2
regr_slope	regr_sxx	regr_sxy	regr_syy	regtypein	regtypeout	regtyperecv
regtypesend	reltime	reltimeeq	reltimege	reltimegt	reltimein	reltimele
reltimelt	reltimene	reltimeout	reltimerecv	reltimesend	repeat	replace

reverse	RI_FKey_cascade_del	RI_FKey_cascade_upd	RI_FKey_check_ins	RI_FKey_check_upd	RI_FKey_noaction_del	RI_FKey_noaction_upd
RI_FKey_restrict_del	RI_FKey_restrict_upd	RI_FKey_setdefault_del	RI_FKey_setdefault_upd	RI_FKey_setnull_del	RI_FKey_setnull_upd	right
round	row_number	row_to_json	rpad	rtrim	scalargtjoinsel	scalargtsel
scalarltjoinsel	scalarltsel	schema_to_xml	schema_to_xml_and_xmlschema	schema_to_xmlschema	session_user	set_bit
set_byte	set_config	set_masklen	shift_jis_2004_to_euc_jis_2004	shift_jis_2004_to_utf8	sjis_to_euc_jp	sjis_to_mic
sjis_to_utf8	smgrin	smgrout	spg_kd_choose	spg_kd_config	spg_kd_inner_consistent	spg_kd_picksplit
spg_quad_choose	spg_quad_config	spg_quad_inner_consistent	spg_quad_leaf_consistent	spg_quad_picksplit	spg_text_choose	spg_text_config
spg_text_inner_consistent	spg_text_leaf_consistent	spg_text_picksplit	spgbeginscan	spgbuild	spgbuildempty	spgbulkdelete
spgcanreturn	spgcostestimate	spgendscan	spggetbitmap	spggettuple	spginsert	spgmarkpos
spgoptions	spgrescan	spgrestrpos	spgvacuumcleanup	stddev	stddev_pop	stddev_samp
string_agg	string_agg_finalfn	string_agg_transfn	strip	sum	suppress_redundant_updates_trigger	table_to_xml

table_to_xml_and_xmlschema	table_to_xmlschema	tan	text	text_ge	text_gt	text_larger
text_le	text_lt	text_pattern_ge	text_pattern_gt	text_pattern_le	text_pattern_lt	text_smaller
textanycat	textcat	texteq	texticlike	texticnlike	texticregexeq	texticregexne
textin	textlike	textne	textnlike	textout	textrecv	textregexeq
textregexne	textsend	thesaurus_init	thesaurus_lexize	tideq	tidge	tidgt
tidin	tidlarger	tidle	tidlt	tidne	tidout	tidrecv
tidsend	tidsmaller	time	time_cmp	time_eq	time_ge	time_gt
time_hash	time_in	time_larger	time_le	time_lt	time_mi_interval	time_mi_time
time_ne	time_out	time_pl_interval	time_recv	time_send	time_smaller	time_transform
timedate_pl	timemi	timepl	timestamp	timestamp_cmp	timestamp_cmp_date	timestamp_cmp_timestamptz
timestamp_eq	timestamp_eq_date	timestamp_eq_timestamptz	timestamp_ge	timestamp_ge_date	timestamp_ge_timestamptz	timestamp_gt
timestamp_gt_date	timestamp_gt_timestamptz	timestamp_hash	timestamp_in	timestamp_larger	timestamp_le	timestamp_le_date
timestamp_le_timestamptz	timestamp_lt	timestamp_lt_date	timestamp_lt_timestamptz	timestamp_mi	timestamp_mi_interval	timestamp_ne
timestamp_ne_date	timestamp_ne_timestamptz	timestamp_out	timestamp_pl_interval	timestamp_recv	timestamp_send	timestamp_smaller

timestamp_sortsupport	timestamp_transform	timestamptypmodin	timestamptypmodout	timestamptz	timestamptz_cmp	timestamptz_cmp_date
timestamptz_cmp_timestamp	timestamptz_eq	timestamptz_eq_date	timestamptz_eq_timestamp	timestamptz_ge	timestamptz_ge_date	timestamptz_ge_timestamp
timestamptz_gt	timestamptz_gt_date	timestamptz_gt_timestamp	timestamptz_in	timestamptz_larger	timestamptz_le	timestamptz_le_date
timestamptz_le_timestamp	timestamptz_lt	timestamptz_lt_date	timestamptz_lt_timestamp	timestamptz_mi	timestamptz_mi_interval	timestamptz_ne
timestamptz_ne_date	timestamptz_ne_timestamp	timestamptz_out	timestamptz_pl_interval	timestamptz_recv	timestamptz_send	timestamptz_smaller
timestamptztypmodin	timestamptztypmodout	timetypmodin	timetypmodout	timetz	timetz_cmp	timetz_eq
timetz_ge	timetz_gt	timetz_hash	timetz_in	timetz_larger	timetz_le	timetz_lt
timetz_mi_interval	timetz_ne	timetz_out	timetz_pl_interval	timetz_recv	timetz_send	timetz_smaller
timetzdate_pl	timetztypmodin	timetztypmodout	timezone (2069)	timezone (1159)	timezone (2037)	timezone (2070)
timezone (1026)	timezone (2038)	tintervalct	tintervaleq	tintervalge	tintervalgt	tintervalin
tintervalle	tintervalleneq	tintervallenge	tintervallengt	tintervallenle	tintervallenlt	tintervallenne
tintervallt	tintervalne	tintervalout	tintervalov	tintervalrecv	tintervalsame	tintervalsend
tintervalstart	to_ascii (1845)	to_ascii (1847)	to_ascii (1846)	trigger_in	trigger_out	ts_match_qv
ts_match_tq	ts_match_tt	ts_match_vq	ts_rank	ts_rank_cd	ts_rewrite	ts_stat

ts_token_type	ts_typanalyze	tsmatchjoinsel	tsmatchsel	tsq_mcontained	tsq_mcontains	tsquery_and
tsquery_cmp	tsquery_eq	tsquery_ge	tsquery_gt	tsquery_le	tsquery_lt	tsquery_ne
tsquery_not	tsquery_or	tsqueryin	tsqueryout	tsqueryrecv	tsquerysend	tsrange
tsrange_subdiff	tstzrange	tstzrange_subdiff	tsvector_cmp	tsvector_concat	tsvector_eq	tsvector_ge
tsvector_gt	tsvector_le	tsvector_lt	tsvector_ne	tsvector_update_trigger	tsvector_update_trigger_column	tsvectorin
tsvectorout	tsvectorrecv	tsvectorsend	txid_current	txid_current_snapshot	txid_snapshot_in	txid_snapshot_out
txid_snapshot_recv	txid_snapshot_send	txid_snapshot_xip	txid_snapshot_xmax	txid_snapshot_xmin	txid_visible_in_snapshot	uhc_to_utf8
unique_key_recheck	unknownin	unknownout	unknownrecv	unknownsend	unnest	utf8_to_big5
utf8_to_euc_cn	utf8_to_euc_jis_2004	utf8_to_euc_jp	utf8_to_euc_kr	utf8_to_euc_tw	utf8_to_gb18030	utf8_to_gbk
utf8_to_iso8859	utf8_to_iso8859_1	utf8_to_johab	utf8_to_koi8r	utf8_to_koi8u	utf8_to_shift_jis_2004	utf8_to_sjis
utf8_to_uhc	utf8_to_win	uuid_cmp	uuid_eq	uuid_ge	uuid_gt	uuid_hash
uuid_in	uuid_le	uuid_lt	uuid_ne	uuid_out	uuid_recv	uuid_send
var_pop	var_samp	varbit	varbit_in	varbit_out	varbit_recv	varbit_send
varbit_transform	varbitcmp	varbiteq	varbitge	varbitgt	varbitle	varbitlt
varbitne	varbittypmodin	varbittypmodout	varchar	varchar_transform	varcharin	varcharout

varcharrecv	varcharsend	varchartypmodin	varchartypmodout	variance	void_in	void_out
void_recv	void_send	win_to_utf8	win1250_to_latin2	win1250_to_mic	win1251_to_iso	win1251_to_koi8r
win1251_to_mic	win1251_to_win866	win866_to_iso	win866_to_koi8r	win866_to_mic	win866_to_win1251	xideq
xideqint4	xidin	xidout	xidrecv	xidsend	xml	xml_in
xml_is_well_formed	xml_is_well_formed_content	xml_is_well_formed_document	xml_out	xml_recv	xml_send	xmlagg
xmlcomment	xmlconcat2	xmlexists	xmlvalidate	pg_notify	-	-

The following table lists the functions used by GaussDB to implement internal system functions. You are not advised to use these functions. If you need to use them, contact Huawei technical support.

- pv_compute_pool_workload()

  Description: Provides the current load information of the cloud acceleration database instance. (Due to specification changes, the current version no longer supports the current feature. Do not use this feature.)

  Return type: record
- locktag_decode(locktag text)

  Description: Parses lock details from **locktag**.

  Return type: text
- smgreq(a smgr, b smgr)

  Description: Compares two smgrs to check whether they are the same.

  Parameter: smgr, smgr

  Return type: Boolean
- smgrne(a smgr, b smgr)

  Description: Checks whether the two smgrs are different.

  Parameter: smgr, smgr

  Return type: Boolean
- xidin4

  Description: Inputs a 4-byte xid.

  Parameter: cstring

  Return type: xid32

- set_hashbucket_info

  Description: Sets hash bucket information.

  Parameter: text

  Return type: Boolean

- hs_concat

  Description: Concatenates two pieces of hstore data.

  Parameter: hstore, hstore

  Return type: hstore

- hs_contained

  Description: Determines whether two hstore data records are included. The return value is of the Boolean type.

  Parameter: hstore, hstore

  Return type: Boolean

- hs_contains

  Description: Determines whether two hstore data records are included. The return value is of the Boolean type.

  Parameter: hstore, hstore

  Return type: Boolean

- hstore

  Description: Converts parameters to the hstore type.

  Parameter: text, text

  Return type: hstore

- hstore_in

  Description: Receives hstore data in string format.

  Parameter: cstring

  Return type: hstore

- hstore_out

  Description: Sends hstore data in string format.

  Parameter: hstore

  Return type: cstring

- hstore_send

  Description: Sends hstore data in bytea format.

  Parameter: hstore

  Return type: bytea

- hstore_to_array

  Description: Sends hstore data in text array format.

  Parameter: hstore

  Return type: text[]

- hstore_to_matrix

Description: Sends hstore data in text array format.

Parameter: hstore

Return type: text[]

- hstore_version_diag

  Description: Sends hstore data in integer array format.

  Parameter: hstore

  Return type: integer

- int1send

  Description: Packs unsigned 1-byte integers into the internal data buffer stream.

  Parameter: tinyint

  Return type: bytea

- isdefined

  Description: Checks whether a specified key exists.

  Parameter: hstore, text

  Return type: Boolean

- listagg

  Description: Specifies aggregate functions of the list type.

  Parameter: smallint, text

  Return type: text

- log_fdw_validator

  Description: Specifies validation functions.

  Parameter: text[], oid

  Return type: void

- nvarchar2typmodin

  Description: Obtains the typmod information of the varchar type.

  Parameter: cstring[]

  Return type: integer

- nvarchar2typmodout

  Description: Obtains the typmod information of the varchar type, constructs a character string, and returns the character string.

  Parameter: integer

  Return type: cstring

- read_disable_conn_file

  Description: Reads forbidden connection files.

  Parameter: nan

  Return type: disconn_mode text, disconn_host text, disconn_port text, local_host text, local_port text, redo_finished text

- regex_like_m

Description: Specifies the regular expression match, which is used to determine whether a character string complies with a specified regular expression.

Parameter: text, text

Return type: Boolean

- update_pgjob

  Description: Updates a job.

  Parameter: bigint, "char", bigint, timestamp without time zone, timestamp without time zone, timestamp without time zone, timestamp without time zone, timestamp without time zone, smallint, text

  Return type: void

- enum_cmp

  Description: Specifies the enumeration comparison function, which is used to determine whether two enumeration classes are equal and determine their relative sizes.

  Parameter: anyenum, anyenum

  Return type: integer

- enum_eq

  Description: Specifies the enumeration comparison function, which is used to implement the equal sign (=).

  Parameter: anyenum, anyenum

  Return type: Boolean

- enum_first

  Description: Returns the first element in the enumeration class.

  Parameter: anyenum

  Return type: anyenum

- enum_ge

  Description: Specifies the enumeration comparison function, which is used to implement the greater-than sign (>) and equal sign (=).

  Parameter: anyenum, anyenum

  Return type: Boolean

- enum_gt

  Description: Specifies the enumeration comparison function, which is used to implement the greater-than sign (>).

  Parameter: anyenum, anyenum

  Return type: Boolean

- enum_in

  Description: Specifies the enumeration comparison function, which is used to determine whether an element is in an enumeration class.

  Parameter: cstring, oid

  Return type: anyenum

- enum_larger

  Description: Specifies the enumeration comparison function, which is used to implement the greater-than sign (>).

  Parameter: anyenum, anyenum

  Return type: anyenum

- enum_last

  Description: Returns the last element in the enumeration class.

  Parameter: anyenum

  Return type: anyenum

- enum_le

  Description: Specifies the enumeration comparison function, which is used to implement the less-than sign (<) and equal sign (=).

  Parameter: anyenum, anyenum

  Return type: Boolean

- enum_lt

  Description: Specifies the enumeration comparison function, which is used to implement the less-than sign (<).

  Parameter: anyenum, anyenum

  Return type: Boolean

- enum_smaller

  Description: Specifies the enumeration comparison function, which is used to implement the less-than sign (<).

  Parameter: anyenum, anyenum

  Return type: Boolean

- node_oid_name

  Description: Not supported.

  Parameter: oid

  Return type: cstring

- pg_buffercache_pages

  Description: Reads data from the shared buffer.

  Parameter: nan

  Return type: bufferid integer, relfilenode oid, bucketid smallint, reltablespace oid, reldatabase oid, relforknumber smallint, relblocknumber bigint, isdirty boolean, usage_count smallint

- pg_check_xidlimit

  Description: Checks whether nextxid is greater than or equal to xidwarnlimit.

  Parameter: nan

  Return type: Boolean

- pg_comm_delay

  Description: Displays the delay status of the communication library of a single DN.

Parameter: nan

Return type: text, text, integer, integer, integer, integer

- pg_comm_recv_stream

  Description: Displays the receiving stream status of all communication libraries on a single DN.

  Parameter: nan

  Return type: text, bigint, text, bigint, integer, integer, integer, text, bigint, integer, integer, integer, bigint, bigint, bigint, bigint, bigint

- pg_comm_send_stream

  Description: Displays the sending stream status of all communication libraries on a single DN.

  Parameter: nan

  Return type: text, bigint, text, bigint, integer, integer, integer, text, bigint, integer, integer, integer, bigint, bigint, bigint, bigint, bigint

- pg_comm_status

  Description: Displays the communication status of a single DN.

  Parameter: nan

  Return type: text, integer, integer, bigint, bigint, bigint, bigint, bigint, integer, integer, integer, integer, integer

- pg_log_comm_status

  Description: Prints some logs on the DN.

  Parameter: nan

  Return type: Boolean

- pg_parse_clog

  Description: Parses clog to obtain the status of xid.

  Parameter: nan

  Return type: xid xid, status text

- pg_pool_ping

  Description: Sets PoolerPing.

  Parameter: Boolean

  Return type: SETOF boolean

- pg_resume_bkp_flag

  Description: Obtains the delay xlong flag for backup and restoration.

  Parameter: slot_name name

  Return type: start_backup_flag boolean, to_delay boolean, ddl_delay_recycle_ptr text, rewind_time text

- psortoptions

  Description: Returns the psort attribute.

  Parameter: text[], boolean

  Return type: bytea

- xideq4

  Description: Compares two values of the xid type to check whether they are the same.

  Parameter: xid32, xid32

  Return type: Boolean

- xideqint8

  Description: Compares values of the xid type and int8 type to check whether they are the same.

  Parameter: xid, bigint

  Return type: Boolean

- xidlt

  Description: Returns whether xid1 < xid2 is true.

  Parameter: xid, xid

  Return type: Boolean

- xidlt4

  Description: Returns whether xid1 < xid2 is true.

  Parameter: xid32, xid32

  Return type: Boolean

11.5.36 Internal Functions

The following functions of GaussDB use internal data types, which cannot be directly called by users.

- Selection rate calculation functions

areajoinsel	areasel	arraycontjoinsel	arraycontsel	contjoinsel	contsel	eqjoinsel
eqsel	iclikejoinsel	iclikesel	icnlikejoinsel	icnlikesel	icregexeqjoinsel	icregexeqsel
icregexnejoinsel	icregexnesel	likejoinsel	likesel	neqjoinsel	neqsel	nlikejoinsel
nlikesel	positionjoinsel	positionsel	regexeqjoinsel	regexeqsel	regexnejoinsel	regexnesel
scalargtjoinsel	scalargtsel	scalarltjoinsel	scalarltsel	tsmatchjoinsel	tsmatchsel	-

- Statistics collection functions

array_typanalyze	range_typanalyze	ts_typanalyze
local_rto_stat	-	-

- Internal functions for sorting

bpchar_sortsupport	bytea_sortsupport	date_sortsupport	numeric_sortsupport	timestamp_sortsupport

- Internal functions for full-text retrieval

dispell_init	dispell_lexize	dsimple_init	dsimple_lexize	dsnowball_init	dsnowball_lexize	dsynonym_init
dsynonym_lexize	gtsquery_compress	gtsquery_consistent	gtsquery_decompress	gtsquery_penalty	gtsquery_picksplit	gtsquery_same
gtsquery_union	ngram_end	ngram_lextype	ngram_start	pound_end	pound_lextype	pound_start
prsd_end	prsd_headline	prsd_lextype	prsd_start	thesaurus_init	thesaurus_lexize	zhprs_end
zhprs_getlexeme	zhprs_lextype	zhprs_start	-	-	-	-

- Internal type processing functions

abstimerecv	euc_jis_2004_to_utf8	int2recv	line_recv	oidvectorrecv_extend	tidrecv	utf8_to_koi8u
anyarray_recv	euc_jp_to_mic	int2vectorrecv	lseg_recv	path_recv	time_recv	utf8_to_shift_jis_2004
array_recv	euc_jp_to_sjis	int4recv	macaddr_recv	pg_node_tree_recv	time_transform	utf8_to_sjis
ascii_to_mic	euc_jp_to_utf8	int8recv	mic_to_ascii	point_recv	timestamp_recv	utf8_to_uhc
ascii_to_utf8	euc_kr_to_mic	internal_out	mic_to_big5	poly_recv	timestamp_transform	utf8_to_win
big5_to_euc_tw	euc_kr_to_utf8	interval_recv	mic_to_euc_cn	pound_nexttoken	timestamptz_recv	uuid_recv
big5_to_mic	euc_tw_to_big5	interval_transform	mic_to_euc_jp	prsd_nexttoken	timetz_recv	varbit_recv
big5_to_utf8	euc_tw_to_mic	iso_to_koi8r	mic_to_euc_kr	range_recv	tintervalrecv	varbit_transform

bit_recv	euc_tw_to_utf8	iso_to_mic	mic_to_euc_tw	rawrecv	tsqueryrecv	varchar_transform
boolrecv	float4recv	iso_to_win1251	mic_to_iso	record_recv	tsvectorrecv	varcharrecv
box_recv	float8recv	iso_to_win866	mic_to_koi8r	regclassrecv	txid_snapshot_recv	void_recv
bpcharrecv	gb18030_to_utf8	iso8859_1_to_utf8	mic_to_latin1	regconfigrecv	uhc_to_utf8	win_to_utf8
btoidsortsupport	gbk_to_utf8	iso8859_to_utf8	mic_to_latin2	regdictionaryrecv	unknownrecv	win1250_to_latin2
bytearecv	gin_extract_tsvector	johab_to_utf8	mic_to_latin3	regoperatorrecv	utf8_to_ascii	win1250_to_mic
byteawithoutorderwithequalcolrecv	gtsvector_compress	json_recv	mic_to_latin4	regoperrecv	utf8_to_big5	win1251_to_iso
cash_recv	gtsvector_consistent	koi8r_to_iso	mic_to_sjis	regprocedurerecv	utf8_to_euc_cn	win1251_to_koi8r
charrecv	gtsvector_decompress	koi8r_to_mic	mic_to_win1250	regprocrecv	utf8_to_euc_jis_2004	win1251_to_mic
cidr_recv	gtsvector_penalty	koi8r_to_utf8	mic_to_win1251	regtyperecv	utf8_to_euc_jp	win1251_to_win866
cidrecv	gtsvector_picksplit	koi8r_to_win1251	mic_to_win866	reltimerecv	utf8_to_euc_kr	win866_to_iso
circle_recv	gtsvector_same	koi8r_to_win866	namerecv	shift_jis_2004_to_euc_jis_2004	utf8_to_euc_tw	win866_to_koi8r
cstring_recv	gtsvector_union	koi8u_to_utf8	ngram_nexttoken	shift_jis_2004_to_utf8	utf8_to_gb18030	win866_to_mic

date_recv	hll_recv	latin1_to_mic	numeric_recv	sjis_to_euc_jp	utf8_to_gbk	win866_to_win1251
domain_recv	hll_trans_recv	latin2_to_mic	numeric_transform	sjis_to_mic	utf8_to_iso8859	xidrecv
euc_cn_to_mic	hstore_recv	latin2_to_win1250	nvarchar2recv	sjis_to_utf8	utf8_to_iso8859_1	xidrecv4
euc_cn_to_utf8	inet_recv	latin3_to_mic	oidrecv	smalldatetime_recv	utf8_to_johab	xml_recv
euc_jis_2004_to_shift_jis_2004	int1recv	latin4_to_mic	oidvectorrecv	textrecv	utf8_to_koi8r	cstore_tid_out
i16toi1	int16	int16_bool	int16eq	int16div	int16ge	int16gt
int16in	int16le	int16lt	int16mi	int16mul	int16ne	int16out
int16pl	int16recv	int16send	numeric_bool	int2vectorin_extend	int2vectorout_extend	int2vectorrecv_extend
int2vectorsend_extend	tdigest_in	tdigest_merge	tdigest_merge_to_one	tdigest_mergep	tdigest_out	-

- Internal functions for aggregation operations

array_agg_finalfn	array_agg_transfn	bytea_string_agg_finalfn	bytea_string_agg_transfn	date_list_agg_noarg2_transfn	date_list_agg_transfn	float4_list_agg_noarg2_transfn
float4_list_agg_transfn	float8_list_agg_noarg2_transfn	float8_list_agg_transfn	int2_list_agg_noarg2_transfn	int2_list_agg_transfn	int4_list_agg_noarg2_transfn	int4_list_agg_transfn
int8_list_agg_noarg2_transfn	int8_list_agg_transfn	interval_list_agg_noarg2_transfn	interval_list_agg_transfn	list_agg_finalfn	list_agg_noarg2_transfn	list_agg_transfn
median_float8_finalfn	median_interval_finalfn	median_transfn	mode_final	numeric_list_agg_noarg2_transfn	numeric_list_agg_transfn	ordered_set_transition

percentile_cont_float8_final	percentile_cont_interval_final	string_agg_finalfn	string_agg_transfn	timestamp_list_agg_noarg2_transfn	timestamp_list_agg_transfn	timestamptz_list_agg_noarg2_transfn
timestamptz_list_agg_transfn	checksumtext_agg_transfn	-	-	-	-	-

- Hash internal functions

hashbeginscan	hashbuild	hashbuildempty	hashbulkdelete	hashcostestimate	hashendscan	hashgetbitmap
hashgettuple	hashinsert	hashmarkpos	hashmerge	hashrescan	hashrestrpos	hashvacuumcleanup
hashvarlena	-	-	-	-	-	-

- Internal functions of the B-tree index

cbtreebuild	cbtreecanreturn	cbtreecostestimate	cbtreegetbitmap	cbtreegettuple	btbeginscan	btbuild
btbuildempty	btbulkdelete	btcanreturn	btcostestimate	btendscan	btfloat4sortsupport	btfloat8sortsupport
btgetbitmap	btgettuple	btinsert	btint2sortsupport	btint4sortsupport	btint8sortsupport	btmarkpos
btmerge	btnamesortsupport	btrescan	btrestrpos	bttextsortsupport	btvacuumcleanup	cbtreeoptions

- Internal functions of the GiST index

gist_box_compress	gist_box_consistent	gist_box_decompress	gist_box_penalty	gist_box_picksplit	gist_box_same	gist_box_union
gist_circle_compress	gist_circle_consistent	gist_point_compress	gist_point_consistent	gist_point_distance	gist_poly_compress	gist_poly_consistent
gistbeginscan	gistbuild	gistbuildempty	gistbulkdelete	gistcostestimate	gistendscan	gistgetbitmap

gistinsert	gistmarkpos	gistmerge	gistrescan	gistrestrpos	gistvacuumcleanup	range_gist_compress
range_gist_decompress	range_gist_penalty	range_gist_picksplit	range_gist_same	range_gist_union	spg_kd_choose	spg_kd_config
spg_kd_picksplit	spg_quad_choose	spg_quad_config	spg_quad_inner_consistent	spg_quad_leaf_consistent	spg_quad_picksplit	spg_text_choose
spg_text_inner_consistent	spg_text_leaf_consistent	spg_text_picksplit	spgbeginscan	spgbuild	spgbuildempty	spgbulkdelete
spgcostestimate	spgendscan	spggetbitmap	spggettuple	spginsert	spgmarkpos	spgmerge
spgrestrpos	spgvacuumcleanup	-	-	-	-	-

- Internal functions of the GIN index

gin_cmp_prefix	gin_extract_tsquery	gin_tsquery_consistent	gin_tsquery_triconsistent	ginarrayconsistent	ginarrayextract	ginarraytriconsistent
ginbeginscan	ginbuild	ginbuildempty	ginbulkdelete	gincostestimate	ginendscan	gingetbitmap
gininsert	ginmarkpos	ginmerge	ginqueryarrayextract	ginrescan	ginrestrpos	ginvacuumcleanup
cginbuild	cgingetbitmap	-	-	-	-	-

- Internal functions of the Psort index

psortbuild	psortcanreturn	psortcostestimate	psortgetbitmap	psortgettuple

- Internal functions of the UBTree index

ubtbeginscan	ubtbuild	ubtbuildempty	ubtbulkdelete	ubtcanreturn
ubtcostestimate	ubtendscan	ubtgetbitmap	ubtgettuple	ubtinsert

ubtmarkpos	ubtmerge	ubtoptions	ubtrescan	ubtrestrpos
ubtvacuumcleanup	-	-	-	-

- plpgsql internal function

  plpgsql_inline_handler

- Set-related internal functions

array_indexby_delete	array_indexby_length	array_integer_deleteidx	array_integer_exists	array_integer_first	array_integer_last
array_integer_next	array_integer_prior	array_varchar_deleteidx	array_varchar_exists	array_varchar_first	array_varchar_last
array_varchar_next	array_varchar_prior	-	-	-	-

- External table-related internal functions

dist_fdw_handler	roach_handler	streaming_fdw_handler	dist_fdw_validator	file_fdw_handler	file_fdw_validator	log_fdw_handler

- Auxiliary function for the primary DN to remotely read the data page from the standby DN.

  gs_read_block_from_remote is used to read the pages of a non-segment-page table file. By default, only the initial user can view the data. Other users can use the data only after being granted with permissions.

  gs_read_segment_block_from_remote is used to read the pages of a segment-page table file. By default, only the initial user can view the data. Other users can use the data only after being granted with permissions.

- Auxiliary function for the primary DN to remotely read the data file from the standby DN.

  The **gs_read_file_from_remote** command is used to read a specified file. After obtaining the file size using the **gs_read_file_size_from_remote** function, **gs_repair_file** reads the remote file segment by segment using this function. By default, only the initial user can view the data. Other users can use the data only after being granted with permissions.

  The **gs_read_file_size_from_remote** command is used to read the size of a specified file. This command is used to read the size of a specified file. Before using the **gs_repair_file** function to repair a file, you need to obtain the size of the file from the remote end to verify the missing file information and repair the missing files one by one. By default, only the initial user can view the data. Other users can use the data only after being granted with permissions.

- AI feature functions

create_snapshot	create_snapshot_internal	prepare_snapshot_internal	prepare_snapshot	manage_snapshot_internal	archive_snapshot	publish_snapshot
purge_snapshot_internal	purge_snapshot	sample_snapshot	-	-	-	-

– Functions of **PKG_SERVICE**

isubmit_on_nodes	submit_on_nodes	-	-	-	-	-

- Other functions

to_tsvector_for_batch	value_of_percentile	disable_conn	bind_variable	job_update	job_cancel	job_finish
similar_escape	table_skewness (unavailable)	timetz_text	time_text	reltime_text	abstime_text	_pg_keysequal
analyze_query (unavailable)	analyze_workload (unavailable)	ssign_table_type	gs_comm_proxy_thread_status	gs_txid_oldestxmin	pg_cancel_session	pg_stat_segment_space_info
remote_segment_space_info	set_cost_params	set_weight_params	start_collect_workload	tdigest_in	tdigest_merge	tdigest_merge_to_one
tdigest_mergep	tdigest_out	pg_get_delta_info	-	-	-	-

- View-related reference functions

  adm_hist_sqlstat_func

  adm_hist_sqlstat_idlog_func

11.5.37 Global SysCache Feature Functions

The current feature is a lab feature. Contact Huawei technical support before using it.

- gs_gsc_table_detail(database_id default NULL, rel_id default NULL)

  Description: Queries global system cache table metadata in a database. The user who calls this function must have the SYSADMIN permission.

  Parameter: Specifies the database and table whose global system cache is to be queried. The default value of **database_id** is **NULL** or **–1**, indicating all databases. The value **0** indicates a shared table. Other values indicate the specified database and shared table. **rel_id** indicates the OID of the specified table. The default value **NULL** or **–1** indicates all tables. Other values indicate the specified table. If the table does not exist, an error is reported. If the OID does not exist, a null result is returned.

  Return type: Tuple

  Example:

```
select * from gs_gsc_table_detail(-1) limit 1;
database_oid | database_name | reloid |         relname         | relnamespace | reltype | reloftype |
relowner | relam | relfilenode | reltablespace | relhasindex | relisshared | relkind | relnatts | relhasoids |
relhaspkey | parttype | tdhasuids | attnames  | extinfo
--------------+---------------+--------+-------------------------+--------------+---------+-----------+----------
+-------+-------------+---------------+-------------+-------------+---------+----------+------------+------------
+----------+-----------+-----------+---------
0 |               |   2676 | pg_authid_rolname_index |           11 |       0 |         0 |       10 |   403 |           0
|         1664 | f           | t           | i       |        1 | f          | f          | n        | f         | 'rolname' |
(1 row)
```

- gs_gsc_catalog_detail(database_id default NULL, rel_id default NULL)

  Description: Queries the system table row information cached in the global system in a database. The user who calls this function must have the SYSADMIN permission.

  Parameter: Specifies the database and table whose global system cache is to be queried. The default value of **database_id** is **NULL** or **–1**, indicating all databases. The value **0** indicates a shared table. Other values indicate the specified database and shared table. **rel_id** indicates the ID of the specified table, including all system catalogs that have system caches. The default value **NULL** or **–1** indicates all tables. Other values indicate the specified table. If the database does not exist, an error is reported. If the table does not exist, a null result is returned.

  Return type: Tuple

  Example:

```
openGauss=#
select * from gs_gsc_catalog_detail(16574, 1260);
 database_id | database_name | rel_id | rel_name  | cache_id |  self  |  ctid  | infomask | infomask2 |
hash_value | refcount
-------------+---------------+--------+-----------+----------+--------+--------+----------+-----------+------------
+----------
           0 |               |   1260 | pg_authid |       10 | (0, 9) | (0, 9) |    10507 |        26 |  531311568 |
10
           0 |               |   1260 | pg_authid |       11 | (0, 4) | (0, 4) |     2313 |        26 |  365368336 |        1
           0 |               |   1260 | pg_authid |       11 | (0, 9) | (0, 9) |    10507 |        26 | 3911517328 |
10
           0 |               |   1260 | pg_authid |       11 | (0, 7) | (0, 7) |     2313 |        26 | 1317799983 |
1
           0 |               |   1260 | pg_authid |       11 | (0, 5) | (0, 5) |     2313 |        26 | 3664347448 |
1
           0 |               |   1260 | pg_authid |       11 | (0, 1) | (0, 1) |     2313 |        26 |  276477273 |        1
           0 |               |   1260 | pg_authid |       11 | (0, 3) | (0, 3) |     2313 |        26 | 2465837659 |
1
           0 |               |   1260 | pg_authid |       11 | (0, 8) | (0, 8) |     2313 |        26 | 3205288035 |
1
           0 |               |   1260 | pg_authid |       11 | (0, 6) | (0, 6) |     2313 |        26 |  131811687 |        1
```

```
          0 |              |   1260 | pg_authid |       11 | (0, 2) | (0, 2) |     2313 |        26 | 1226484587 |
1
(10 rows)
```

- gs_gsc_clean(database_id default NULL)

  Description: Clears the global syscache cache. Note that data in use will not be cleared. The user who calls this function must have the SYSADMIN permission.

  Parameter: Specifies the database whose global system cache needs to be cleared. The default value **NULL** or **–1** indicates that the global system cache of all databases is cleared. The value **0** indicates that the global system cache of only the shared table is cleared. Other values indicate that the global system cache of the specified database and shared table is cleared. If the database does not exist, an error is reported.

  Return type: Boolean

  Example:

```
openGauss=# select * from gs_gsc_clean();
gs_gsc_clean
--------------
t
(1 row)
```

- gs_gsc_dbstat_info(database_id default NULL)

  Description: Obtains GSC memory statistics on the local node, including cache query, hit, loading, expiration, and occupied space information of tuples, relationships, and partitions, database-level elimination information, thread reference information, and memory usage information. This parameter can be used to locate performance problems. For example, if the value of the hits/searches array is far less than 1, the value of **global_syscache_threshold** may be too small. As a result, the query hit ratio decreases. The user who calls this function must have the SYSADMIN permission.

  Parameter: Specifies the global system cache statistics of the database to be queried. **NULL** or **–1** indicates that all databases are queried. **0** indicates that only the information about the shared table is queried. Other values indicate that the information about the specified database and shared table is queried. Invalid input value. If the database does not exist, an error is reported.

  Return type: Tuple

  Example:

```
openGauss=# select * from gs_gsc_dbstat_info();
database_id | database_name | tup_searches | tup_hits | tup_miss | tup_count | tup_dead | tup_memory
| rel_searches | rel_hits | rel_mis
s | rel_count | rel_dead | rel_memory | part_searches | part_hits | part_miss | part_count | part_dead |
part_memory | total_memory | swa
pout_count | refcount
-------------+---------------+--------------+----------+----------+-----------+----------+------------
+--------------+----------+--------
--+-----------+----------+------------+---------------+-----------+-----------+------------+-----------
+-------------+--------------+----
-----------+----------
          0 |              |          300 |      235 |       31 |        22 |        2 |       9752 |          598 |      108
|        1
8 |        18 |        0 |      77720 |             0 |         0 |         0 |          0 |         0 |           0 |       752912 |
          0 |        0
      16574 | postgres      |         3368 |     2289 |      329 |       273 |        0 |      92593 |         1113 |
524 |        4
8 |        48 |        0 |     340456 |             0 |         0 |         0 |          0 |         0 |           0 |      4124792 |
          0 |       10
(2 rows)
```

11.5.38 Data Damage Detection and Repair Functions

- gs_verify_data_file(verify_segment bool)

  Description: Checks whether files in the current database of the current instance are lost. The verification only checks whether intermediate segments are lost in the main file of the data table. The default value is **false**, indicating that the segment-page table data file is not verified. If this parameter is set to **true**, only segment-page table files are verified. By default, only initial users, users with the **sysadmin** permission, and users with the O&M administrator attribute in the O&M mode can view the information. Other users can view the information only after being granted with permissions.

  The returned result is as follows:

  – Non-segment-page table: **rel_oid** and **rel_name** indicate the table OID and table name of the corresponding file, and **miss_file_path** indicates the relative path of the lost file.

  – Segment-paged table: All tables are stored in the same file. Therefore, **rel_oid** and **rel_name** cannot display information about a specific table. For a segment-page table, if the first file is damaged, the subsequent files such as .1 and .2 are not checked. For example, if 3, 3.1, and 3.2 are damaged, only 3 damage can be detected. When the number of segment-page files is less than 5, the files that are not generated are also detected during function detection. For example, if there are only files 1 and 2, files 3, 4, and 5 are detected during segment-page file detection. In the following examples, the first is an example of checking a non-segment-page table, and the second is an example of checking a segment-page table.

  Parameter description:

  – verify_segment

    Specifies the range of files to be checked. **false** indicates that non-segment-page tables are verified. **true** indicates that segment-page tables are verified.

    The value can be **true** or **false** (default value).

  Return type: record

  Example:

  Verify a non-segment-page table.

```
openGauss=# select * from gs_verify_data_file();
node_name         | rel_oid |  rel_name    |  miss_file_path
------------------+---------+--------------+------------------
dn_6001_6002_6003 |   16554 |     test     | base/16552/24745
```

  Verify a segment-page table.

```
openGauss=# select * from gs_verify_data_file(true);
     node_name     | rel_oid | rel_name | miss_file_path
-------------------+---------+----------+----------------
 dn_6001_6002_6003 |       0 | none     | base/16573/2
```

- gs_repair_file(tableoid Oid, path text, timeout int)

  Description: Repairs the file based on the input parameters. Only the primary DN with normal primary/standby connection is supported. The parameter is set based on the OID and path returned by the **gs_verify_data_file** function. The value of table OID for a segment-page table ranges from 0 to 4294967295. (The internal verification determines whether a file is a segment-page table file based on the file path. The table OID is not used for a segment-page table file.) If the repair is successful, **true** is returned. If the repair fails, the failure cause is displayed. By default, only the initial user, users with the **sysadmin** permission, and users with the O&M administrator attribute in O&M mode on the primary DN can view the table. Other users can view the table only after being granted with permissions.

CAUTION

1. If a file on a DN is damaged, a verification error at the PANIC level is reported when the DN is promoted to primary. The DN cannot be promoted to primary, which is normal.
2. If a file exists but its size is 0, the file will not be repaired. To repair the file, you need to delete the file whose size is 0 and then repair it.
3. You can delete a file only after the file descriptor is automatically closed. You can manually restart the process or perform a primary/standby switchover.

Parameter description:

– tableoid

  OID of the table corresponding to the file to be repaired. Set this parameter based on the **rel_oid** column in the list returned by the **gs_verify_data_file** function.

  Value range: OID ranging from 0 to 4294967295. Note: A negative value will be forcibly converted to a non-negative integer.

– path

  Path of the file to be repaired. Set this parameter based on the **miss_file_path** column in the list returned by the **gs_verify_data_file** function.

  Value range: a string

– timeout

  Specifies the duration for waiting for the standby DN to replay. The repair file needs to wait for the standby DN to be put back to the corresponding location on the current primary DN. Set this parameter based on the replay duration of the standby DN.

  Value range: 60s to 3600s.

Return type: Boolean

Example:

```
openGauss=# select * from gs_repair_file(16554,'base/16552/24745',360);
gs_repair_file
----------------
t
```

- local_bad_block_info()

  Description: Displays the page damage of the instance. You can read the page from the disk and record the page CRC failure. By default, only initial users, users with the **sysadmin** permission, users with the monitoring administrator attribute, users with the O&M administrator attribute in the O&M mode, and monitoring users can view the information. Other users can view the information only after being granted with permissions.

In the displayed information, **file_path** indicates the relative path of the damaged file. If the table is a segment-page table, the logical information instead of the actual physical file information is displayed. **block_num** indicates the number of the page where the file is damaged. The page number starts from 0. **check_time** indicates the time when the page damage is detected. **repair_time** indicates the time when the page is repaired.

Return type: record

Example:

```
openGauss=# select * from local_bad_block_info();
node_name    | spc_node | db_node | rel_node| bucket_node | fork_num | block_num |    file_path     |
check_time            |   repair_time
-----------------+--------+--------+--------+--------------+----------+-----------+-----------------
+--------------------------+-------------------------------
dn_6001_6002_6003|  1663 |  16552 |  24745 |        -1    |    0    | 0        | base/16552/24745 |
2022-01-13 20:19:08.385004+08 | 2022-01-13 20:19:08.407314+08
```

- local_clear_bad_block_info()

  Description: Deletes data of repaired pages from **local_bad_block_info**, that is, information whose **repair_time** is not empty. By default, only initial users, users with the **sysadmin** permission, users with the O&M administrator attribute in the O&M mode, and monitoring users can view the information. Other users can view the information only after being granted with permissions.

  Return type: Boolean

  Example:

```
openGauss=# select * from local_clear_bad_block_info();
result
--------
t
```

- gs_verify_and_tryrepair_page (path text, blocknum oid, verify_mem bool, is_segment bool)

  Description: Verifies the page specified by the instance. By default, only the initial user, users with the **sysadmin** permission, and users with the O&M administrator attribute in O&M mode on the primary DN can view the table. Other users can view the table only after being granted with permissions.

  In the command output, **disk_page_res** indicates the verification result of the page on the disk, **mem_page_res** indicates the verification result of the page in the memory, and **is_repair** indicates whether the repair function is triggered during the verification. **t** indicates that the page is repaired, and **f** indicates that the page is not repaired.

  Note: If a page on a DN is damaged, a verification error at the PANIC level is reported when the DN is promoted to primary. The DN cannot be promoted to primary, which is normal. Damaged pages of hash bucket tables cannot be repaired.

  Parameter description:

  – path

    Path of the damaged file. Set this parameter based on the **file_path** column in the **local_bad_block_info** file.

    Value range: a string

  - blocknum

    Page number of the damaged file. Set this parameter based on the **block_num** column in the **local_bad_block_info** file.

    Value range: OID ranging from 0 to 4294967295. Note: A negative value will be forcibly converted to a non-negative integer.

  - verify_mem

    Specifies whether to verify a specified page in the memory. If this parameter is set to **false**, only pages on the disk are verified. If this parameter is set to **true**, pages in the memory and on the disk are verified. If a page on the disk is damaged, the system verifies the basic information of the page in the memory and flushes the page to the disk to restore the page. If a page is not found in the memory during memory page verification, the page on the disk is read through the memory API. During this process, if the disk page is faulty, the remote read automatic repair function is triggered.

    Value range: The value is of a Boolean type and can be **true** or **false**.

  - is_segment

    Determines whether the table is a segment-page table. Set this parameter based on the value of **bucket_node** in the **local_bad_block_info** file. If the value of **bucket_node** is **–1**, the table is not a segment-page table. In this case, set **is_segment** to **false**. If the value of **bucket_node** is not –1, set **is_segment** to **true**.

    Value range: The value is of a Boolean type and can be **true** or **false**.

  Return type: record

  Example:

```
openGauss=# select * from gs_verify_and_tryrepair_page('base/16552/24745',0,false,false);
node_name        |       path       |  blocknum  |        disk_page_res        | mem_page_res | is_repair
------------------+------------------+------------+-----------------------------+---------------+----------
dn_6001_6002_6003 | base/16552/24745 |     0      | page verification succeeded.|              | f
```

- gs_repair_page(path text, blocknum oid, is_segment bool, timeout int)

  Description: Restores the specified page of the instance. This function can be used only by the primary DN that is properly connected to the primary and standby DNs. If the page is successfully restored, **true** is returned. If an error occurs during the restoration, an error message is displayed. By default, only the initial user, users with the **sysadmin** permission, and users with the O&M administrator attribute in O&M mode on the primary DN can view the table. Other users can view the table only after being granted with permissions.

  Note: If a page on a DN is damaged, a verification error at the PANIC level is reported when the DN is promoted to primary. The DN cannot be promoted to primary, which is normal. Damaged pages of hash bucket tables cannot be repaired.

  Parameter description:

  - path

    Path of the damaged page. Set this parameter based on the **file_path** column in **local_bad_block_info** or the **path** column in the **gs_verify_and_tryrepair_page** function.

    Value range: a string

  - blocknum

    Number of the damaged page. Set this parameter based on the **block_num** column in **local_bad_block_info** or the **blocknum** column in the **gs_verify_and_tryrepair_page** function.

    Value range: OID ranging from 0 to 4294967295. Note: A negative value will be forcibly converted to a non-negative integer.

  - is_segment

    Determines whether the table is a segment-page table. The value of this parameter is determined by the value of **bucket_node** in **local_bad_block_info**. If the value of **bucket_node** is **–1**, the table is not a segment-page table and **is_segment** is set to **false**. If the value of **bucket_node** is not –1, **is_segment** is set to true.

    Value range: The value is of a Boolean type and can be **true** or **false**.

  - timeout

    Duration of waiting for standby DN replay. The page to be repaired needs to wait for the standby DN to be played back to the location of the current primary DN. Set this parameter based on the playback duration of the standby DN.

    Value range: 60s to 3600s.

  Return type: Boolean

  Example:

```
openGauss=# select * from gs_repair_page('base/16552/24745',0,false,60);
result
--------
t
```

- gs_verify_urq(index_oid oid, queue_type text, blocknum bigint, verify_type text)

  Description: Verifies the correctness of the index recycling queue or the performance of obtaining index pages from the recycling queue.

  Parameter description:

  - index_oid (UB-tree index OID)
  - queue_type (queue type)

    empty queue: potential queue.

    free queue: available queue.

    single page: single page of the queue

  - blocknum (page number)

    If the queue type is single page, all tuples of **blocknum** on a single page are verified. The value range is [0,Queue file size/8192).

    If the queue type is empty queue or free queue and **blocknum** is not set to **0**, all tuples on all pages of this queue are verified. If **blocknum** is set to **0**, page tuples are not verified.

  - verify_type (verification type)

    physics verifies the correctness of the physical structure of the queue.

    performance: verifies the performance of obtaining pages from the recycling queue.

  Return type: record

Table 11-91 gs_verify_urq parameters

Category	Parameter	Type	Description
Input parameter	index_oid	oid	UB-tree index OID.
Input parameter	queue_type	text	Queue type. ● **empty queue**: potential queue ● **free queue**: available queue ● **single page**: single page of the queue
Input parameter	blocknum	bigint	Page number.
Input parameter	verify_type	text	Specifies the verification type: ● **physics**: verifies the physical structure of a column. ● **performance**: verifies the performance of obtaining pages from the recycling queue.
Output parameter	verify_code	text	Error code
Output parameter	detail	text	Error description

Example:

```
openGauss=# select * from gs_verify_urq(16387,'free queue',1,'physics');
 verify_code | detail
-------------+--------
```

NOTE

Currently, this interface only supports USTORE index tables and does not support partition local indexes.

- gs_repair_urq(index_oid oid)

Description: Recreates an index to recycle queues (potential and available queues). If the repair is successful, **reinitial the recycle queue of index relation successfully** is displayed.

Parameter description:

– index_oid (UB-tree index OID)

Return type: text

Example:

```
openGauss=# select * from gs_repair_urq(16387);
                          result
-------------------------------------------------------------
 reinitial the recycle queue of index relation sucessfully.
```

NOTE

Currently, this interface only supports USTORE index tables and does not support partition local indexes.

11.5.39 Obsolete Functions

The following functions in GaussDB have been discarded in the latest version:

gs_wlm_get_session_info	gs_wlm_get_user_session_info	pgxc_get_csn	pgxc_get_stat_dirty_tables	pgxc_get_thread_wait_status	pgxc_gtm_snapshot_status	pgxc_is_committed
pgxc_lock_for_backup	pgxc_lock_for_sp_database	pgxc_lock_for_transfer	pgxc_log_comm_status	pgxc_max_datanode_size	pgxc_node_str	pgxc_pool_check
pgxc_pool_connection_status	pgxc_pool_reload	pgxc_prepared_xact	pgxc_snapshot_status	pgxc_stat_dirty_tables	pgxc_unlock_for_sp_database	pgxc_unlock_for_transfer
pgxc_version	array_extend	prepare_statement_status	remote_rto_stat	dbe_perf.global_slow_query_info	dbe_perf.global_slow_query_info_bytime	dbe_perf.global_slow_query_history

pg_stat_get_pooler_status	pg_stat_get_wlm_node_resource_info	pg_stat_get_wlm_session_info_internal	DBE_PERF.get_wlm_controlgroup_ng_config()	DBE_PERF.get_wlm_user_resource_runtime()	global_space_shrink	pg_pool_validate
gs_stat_ustore	table_skewness(text)	table_skewness(text, text, text)	-	-	-	-

11.6 Expressions

11.6.1 Simple Expressions

Logical Expressions

Logical Operators lists the operators and calculation rules of logical expressions.

Comparative Expressions

Comparison Operators lists the common comparative operators.

In addition to comparative operators, you can also use the following sentence structure:

- BETWEEN operator

  a BETWEEN x AND y is equivalent to **a >= x AND a <= y**.

  a NOT BETWEEN x AND y is equivalent to **a < x OR a > y**.

- To check whether a value is null, use:

  expression IS NULL

  expression IS NOT NULL

  or an equivalent (non-standard) sentence structure:

  expression ISNULL

  expression NOTNULL

NOTICE

Do not write **expression=NULL** or **expression<>(!=)NULL**, because **NULL** represents an unknown value, and these expressions cannot determine whether two unknown values are equal.

- is distinct from/is not distinct from
  - is distinct from

    If the data types and values of A and B are different, the value is **true**.

    If the data types and values of A and B are the same, the value is **false**.

    Empty values are considered the same.
  - is not distinct from

    If the data types and values of A and B are different, the value is **false**.

    If the data types and values of A and B are the same, the value is **true**.

    Empty values are considered the same.

Pseudocolumn

ROWNUM

ROWNUM is a pseudocolumn that returns a number indicating the row number of the result obtained from the query. The value of **ROWNUM** in the first row is **1**, the value of **ROWNUM** in the second row is **2**, and so on.

The return type of **ROWNUM** is BIGINT. **ROWNUM** can be used to limit the total number of rows returned by a query. For example, the following statement limits the maximum number of records returned from the table **Students** to 10.

```
select * from Students where rownum <= 10;
```

Examples

```
openGauss=# SELECT 2 BETWEEN 1 AND 3 AS RESULT;
 result
----------
 t
(1 row)

openGauss=# SELECT 2 >= 1 AND 2 <= 3 AS RESULT;
 result
----------
 t
(1 row)

openGauss=# SELECT 2 NOT BETWEEN 1 AND 3 AS RESULT;
 result
----------
 f
(1 row)

openGauss=# SELECT 2 < 1 OR 2 > 3 AS RESULT;
 result
----------
 f
(1 row)

openGauss=# SELECT 2+2 IS NULL AS RESULT;
 result
----------
 f
(1 row)

openGauss=# SELECT 2+2 IS NOT NULL AS RESULT;
 result
----------
 t
(1 row)
```

```
openGauss=# SELECT 2+2 ISNULL AS RESULT;
 result
----------
 f
(1 row)

openGauss=# SELECT 2+2 NOTNULL AS RESULT;
 result
----------
 t
(1 row)

openGauss=# SELECT 2+2 IS DISTINCT FROM NULL AS RESULT;
 result
----------
 t
(1 row)

openGauss=# SELECT 2+2 IS NOT DISTINCT FROM NULL AS RESULT;
 result
----------
 f
(1 row)
```

11.6.2 Condition Expressions

Data that meets the requirements specified by conditional expressions are filtered during SQL statement execution.

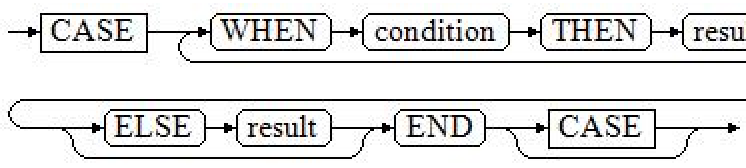
Conditional expressions include the following types:

- CASE

  CASE expressions are similar to the **CASE** statements in other coding languages.

  Figure 11-1 shows the syntax of a **CASE** expression.

  Figure 11-1 case::=



  A **CASE** clause can be used in a valid expression. **condition** is an expression that returns a value of Boolean type.

  – If the result is true, the result of the **CASE** expression is the required result.

  – If the result is false, the following **WHEN** or **ELSE** clauses are processed in the same way.

  – If every **WHEN condition** is false, the result of the expression is the result of the **ELSE** clause. If the **ELSE** clause is omitted and has no match condition, the result is NULL.

  Example:

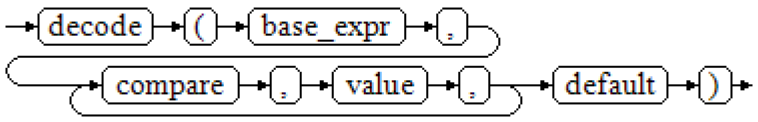
```
openGauss=# CREATE TABLE tpcds.case_when_t1(CW_COL1 INT);
```

```
openGauss=# INSERT INTO tpcds.case_when_t1 VALUES (1), (2), (3);

openGauss=# SELECT * FROM tpcds.case_when_t1;
cw_col1
---------
 1
 2
 3
(3 rows)

openGauss=# SELECT CW_COL1, CASE WHEN CW_COL1=1 THEN 'one' WHEN CW_COL1=2 THEN
'two' ELSE 'other' END FROM tpcds.case_when_t1 ORDER BY 1;
 cw_col1 | case
---------+-------
       1 | one
       2 | two
       3 | other
(3 rows)

openGauss=# DROP TABLE tpcds.case_when_t1;
```

- DECODE

  Figure 11-2 shows the syntax of a **DECODE** expression.

  Figure 11-2 decode::=



  Compare each following **compare(n)** with **base_expr**. **value(n)** is returned if a **compare(n)** matches the **base_expr** expression. If **base_expr** does not match each **compare(n)**, the default value is returned.

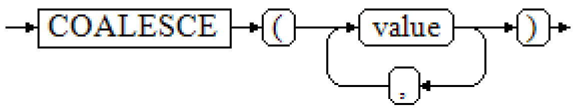
  Conditional Expression Functions describes the examples.

```
openGauss=# SELECT DECODE('A','A',1,'B',2,0);
 case
------
    1
(1 row)
```

- COALESCE

  Figure 11-3 shows the syntax of a **COALESCE** expression.

  Figure 11-3 coalesce::=



  COALESCE returns its first not-**NULL** value. If all the parameters are **NULL**, **NULL** is returned. This value is replaced by the default value when data is displayed. Like a **CASE** expression, **COALESCE** only evaluates the parameters that are needed to determine the result. That is, parameters to the right of the first non-null parameter are not evaluated.

  Example:

```
openGauss=# CREATE TABLE tpcds.c_tabl(description varchar(10), short_description varchar(10), 
last_value varchar(10)) ;

openGauss=# INSERT INTO tpcds.c_tabl VALUES('abc', 'efg', '123');
openGauss=# INSERT INTO tpcds.c_tabl VALUES(NULL, 'efg', '123');

openGauss=# INSERT INTO tpcds.c_tabl VALUES(NULL, NULL, '123');

openGauss=# SELECT description, short_description, last_value, COALESCE(description, 
short_description, last_value) FROM tpcds.c_tabl ORDER BY 1, 2, 3, 4;
 description | short_description | last_value | coalesce 
-------------+-------------------+------------+----------
 abc         | efg               | 123        | abc
             | efg               | 123        | efg
             |                   | 123        | 123
(3 rows)

openGauss=# DROP TABLE tpcds.c_tabl;
```

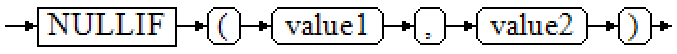
If **description** is not **NULL**, the value of **description** is returned. Otherwise, parameter **short_description** is calculated. If **short_description** is not **NULL**, the value of **short_description** is returned. Otherwise, parameter **last_value** is calculated. If **last_value** is not **NULL**, the value of **last_value** is returned. Otherwise, **none** is returned.

```
openGauss=# SELECT COALESCE(NULL,'Hello World');
   coalesce
---------------
 Hello World
(1 row)
```

- NULLIF

  Figure 11-4 shows the syntax of a **NULLIF** expression.

  Figure 11-4 nullif::=



Only if **value1** is equal to **value2** can **NULLIF** return the **NULL** value. Otherwise, **value1** is returned.

Example:

```
openGauss=# CREATE TABLE tpcds.null_if_t1 (
    NI_VALUE1 VARCHAR(10),
    NI_VALUE2 VARCHAR(10)
);

openGauss=# INSERT INTO tpcds.null_if_t1 VALUES('abc', 'abc');
openGauss=# INSERT INTO tpcds.null_if_t1 VALUES('abc', 'efg');

openGauss=# SELECT NI_VALUE1, NI_VALUE2, NULLIF(NI_VALUE1, NI_VALUE2) FROM tpcds.null_if_t1 
ORDER BY 1, 2, 3;

 ni_value1 | ni_value2 | nullif 
-----------+-----------+--------
 abc       | abc       | 
 abc       | efg       | abc
(2 rows)
openGauss=# DROP TABLE tpcds.null_if_t1;
```

If the value of **value1** is equal to that of **value2**, **NULL** is returned. Otherwise, the value of **value1** is returned.

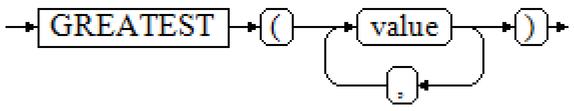
```
openGauss=# SELECT NULLIF('Hello','Hello World');
 nullif
```

```
--------
 Hello
(1 row)
```

- GREATEST (maximum value) and LEAST (minimum value)

  Figure 11-5 shows the syntax of a **GREATEST** expression.

  Figure 11-5 greatest::=

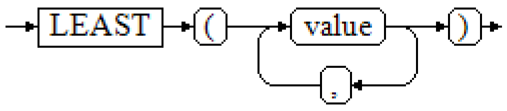


  You can select the maximum value from any numerical expression list.

```
openGauss=# SELECT greatest(9000,155555,2.01);
 greatest
----------
   155555
(1 row)
```

  Figure 11-6 shows the syntax of a **LEAST** expression.

  Figure 11-6 least::=



  You can select the minimum value from any numerical expression list.

  Each of the preceding numeric expressions can be converted into a common data type, which will be the data type of the result.

  The NULL values in the list will be ignored. The result is **NULL** only if the results of all expressions are **NULL**.

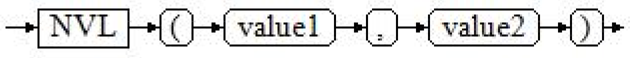
```
openGauss=# SELECT least(9000,2);
 least
-------
     2
(1 row)
```

  Conditional Expression Functions describes the examples.

- NVL

  Figure 11-7 shows the syntax of an **NVL** expression.

  Figure 11-7 nvl::=



  If the value of **value1** is **NULL**, the value of **value2** is returned. Otherwise, the value of **value1** is returned.

  Example:

```
openGauss=# SELECT nvl(null,1);
nvl
-----
 1
(1 row)
openGauss=# SELECT nvl ('Hello World' ,1);
     nvl
---------------
 Hello World
(1 row)
```

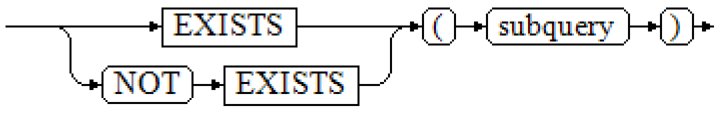
11.6.3 Subquery Expressions

Subquery expressions include the following types:

- EXISTS/NOT EXISTS

  Figure 11-8 shows the syntax of an **EXISTS/NOT EXISTS** expression.

  Figure 11-8 EXISTS/NOT EXISTS::=



  The parameter of an **EXISTS** expression is an arbitrary **SELECT** statement, or subquery. The subquery is evaluated to determine whether it returns any rows. If it returns at least one row, the result of **EXISTS** is "true". If the subquery returns no rows, the result of **EXISTS** is "false".

  The subquery will generally only be executed long enough to determine whether at least one row is returned, not all the way to completion.

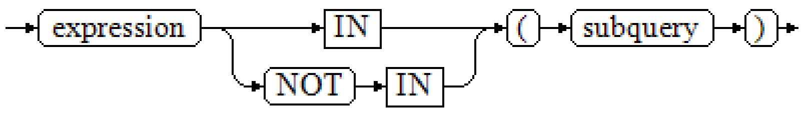
  Example:

```
openGauss=# SELECT sr_reason_sk,sr_customer_sk FROM tpcds.store_returns WHERE EXISTS (SELECT
d_dom FROM tpcds.date_dim WHERE d_dom = store_returns.sr_reason_sk and sr_customer_sk <10);
sr_reason_sk | sr_customer_sk
--------------+----------------
           13 |              2
           22 |              5
           17 |              7
           25 |              7
            3 |              7
           31 |              5
            7 |              7
           14 |              6
           20 |              4
            5 |              6
           10 |              3
            1 |              5
           15 |              2
            4 |              1
           26 |              3
(15 rows)
```

- IN/NOT IN

  Figure 11-9 shows the syntax of an **IN/NOT IN** expression.

Figure 11-9 IN/NOT IN::=



The right-hand side is a parenthesized subquery, which must return exactly one column. The left-hand expression is evaluated and compared to each row of the subquery result. The result of **IN** is "true" if any equal subquery row is found. The result is "false" if no equal row is found (including the case where the subquery returns no rows).

This is in accordance with SQL normal rules for Boolean combinations of null values. If the columns corresponding to two rows equal and are not empty, the two rows are equal to each other. If any columns corresponding to the two rows do not equal and are not empty, the two rows are not equal to each other. Otherwise, the result is **NULL**. If there are no equal right-hand values and at least one right-hand row yields null, the result of **IN** will be null, not false.

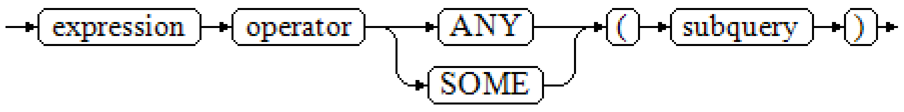
Example:

```
openGauss=# SELECT sr_reason_sk,sr_customer_sk  FROM tpcds.store_returns WHERE sr_customer_sk
IN (SELECT d_dom FROM tpcds.date_dim WHERE d_dom < 10);
sr_reason_sk | sr_customer_sk
--------------+----------------
          10 |              3
          26 |              3
          22 |              5
          31 |              5
           1 |              5
          32 |              5
          32 |              5
           4 |              1
          15 |              2
          13 |              2
          33 |              4
          20 |              4
          33 |              8
           5 |              6
          14 |              6
          17 |              7
           3 |              7
          25 |              7
           7 |              7
(19 rows)
```

- ANY/SOME

  Figure 11-10 shows the syntax of an **ANY/SOME** expression.

Figure 11-10 any/some::=



The right-hand side is a parenthesized subquery, which must return exactly one column. The left-hand expression is evaluated and compared to each row of the subquery result using the given operator, which must yield a Boolean result. The result of **ANY** is "true" if any true result is obtained. The result is "false" if no true result is found (including the case where the subquery returns no rows). **SOME** is a synonym of **ANY**. **IN** can be equivalently replaced with **ANY**.

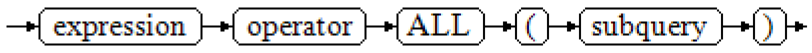
Example:

```
openGauss=# SELECT sr_reason_sk,sr_customer_sk  FROM tpcds.store_returns WHERE sr_customer_sk
< ANY (SELECT d_dom FROM tpcds.date_dim WHERE d_dom < 10);
sr_reason_sk | sr_customer_sk
--------------+----------------
          26 |              3
          17 |              7
          32 |              5
          32 |              5
          13 |              2
          31 |              5
          25 |              7
           5 |             6
           7 |             7
          10 |              3
           1 |             5
          14 |              6
           4 |             1
           3 |             7
          22 |              5
          33 |              4
          20 |              4
          33 |              8
          15 |              2
(19 rows)
```

- ALL

  Figure 11-11 shows the syntax of an **ALL** expression.

  Figure 11-11 all::=



The right-hand side is a parenthesized subquery, which must return exactly one column. The left-hand expression is evaluated and compared to each row of the subquery result using the given operator, which must yield a Boolean result. The result of **ALL** is "true" if all rows yield true (including the case where the subquery returns no rows). The result is "false" if any false result is found.

Example:

```
openGauss=# SELECT sr_reason_sk,sr_customer_sk  FROM tpcds.store_returns WHERE sr_customer_sk
< all(SELECT d_dom FROM tpcds.date_dim WHERE d_dom < 10);
 sr_reason_sk | sr_customer_sk
--------------+----------------
(0 rows)
```

11.6.4 Array Expressions

IN

expression **IN** *(value [, ...])*

The parentheses on the right contain an expression list. The expression result on the left is compared with the content in the expression list. If the content in the list meets the expression result on the left, the result of **IN** is **true**. If no result meets the requirements, the result of **IN** is **false**.

Example:

```
openGauss=# SELECT 8000+500 IN (10000, 9000) AS RESULT;
  result
----------
 f
(1 row)
```

If the expression result is null or the expression list does not meet the expression conditions and at least one empty value is returned for the expression list on the right, the result of **IN** is **null** rather than **false**. This method is consistent with the Boolean rules used when SQL statements return empty values.

NOT IN

expression **NOT IN** *(value [, ...])*

The parentheses on the right contain an expression list. The expression result on the left is compared with the content in the expression list. If the content in the list does not meet the expression result on the left, the result of **NOT IN** is **true**. If any content meets the expression result, the result of **NOT IN** is **false**.

Example:

```
openGauss=# SELECT 8000+500 NOT IN (10000, 9000) AS RESULT;
  result
----------
 t
(1 row)
```

If the query statement result is null or the expression list does not meet the expression conditions and at least one empty value is returned for the expression list on the right, the result of **NOT IN** is **null** rather than **false**. This method is consistent with the Boolean rules used when SQL statements return empty values.

NOTE

In all situations, **X NOT IN Y** equals to **NOT(X IN Y)**.

ANY/SOME (array)

expression operator **ANY** *(array expression)*

expression operator **SOME** *(array expression)*

```
openGauss=# SELECT 8000+500 < SOME (array[10000,9000]) AS RESULT;
  result
----------
 t
(1 row)
openGauss=# SELECT 8000+500 < ANY (array[10000,9000]) AS RESULT;
  result
----------
 t
(1 row)
```

The right-hand side is a parenthesized expression, which must yield an array value. The result of the expression on the left uses operators to compute and compare the results in each row of the array expression. The comparison result must be a Boolean value.

- If at least one comparison result is true, the result of **ANY** is **true**.
- If no comparison result is true, the result of ANY is false.
- If no comparison result is true and the array expression generates at least one null value, the value of ANY is NULL, rather than false. This method is consistent with the Boolean rules used when SQL statements return empty values.

NOTE

SOME is a synonym of **ANY**.

ALL (array)

expression operator **ALL** *(array expression)*

The right-hand side is a parenthesized expression, which must yield an array value. The result of the expression on the left uses operators to compute and compare the results in each row of the array expression. The comparison result must be a Boolean value.

- The result of **ALL** is **true** if all comparisons yield **true** (including the case where the array has zero elements).
- The result of **ALL** is **false** if one or multiple comparisons yield **false**.
- If the array expression yields a null array, the result of **ALL** will be null. If the left-hand expression yields null, the result of **ALL** is ordinarily null (though a non-strict comparison operator could possibly yield a different result). Also, if the right-hand array contains any null elements and no false comparison result is obtained, the result of **ALL** will be null, not true (again, assuming a strict comparison operator). This method is consistent with the Boolean rules used when SQL statements return empty values.

```
openGauss=# SELECT 8000+500 < ALL (array[10000,9000]) AS RESULT;
 result
----------
 t
(1 row)
```

11.6.5 Row Expressions

The syntax is as follows:

```
row_constructor operator row_constructor
```

Both sides of the row expression are row constructors. The values of both rows must have the same number of fields and they are compared with each other. The row comparison allows operators including =, <>, <, <=, and >= or a similar operator.

The use of operators =<> is slightly different from other operators. If all fields of two rows are not empty and equal, the two rows are equal. If any field in two rows is not empty and not equal, the two rows are not equal. Otherwise, the comparison result is null.

For operators <, <=, >, and > =, the fields in rows are compared from left to right until a pair of fields that are not equal or are empty are detected. If the pair of fields contains at least one null value, the comparison result is null. Otherwise, the comparison result of this pair of fields is the final result.

Example:

```
openGauss=# SELECT ROW(1,2,NULL) < ROW(1,3,0) AS RESULT;
  result
----------
 t
(1 row)
```

11.7 Type Conversion

11.7.1 Overview

Background

SQL is a typed language. That is, every data item has an associated data type which determines its behavior and allowed usage. GaussDB has an extensible type system that is more general and flexible than other SQL implementations. Hence, most type conversion behaviors in GaussDB are governed by general rules. This allows the use of mixed-type expressions.

The GaussDB scanner/parser divides lexical elements into five fundamental categories: integers, floating-point numbers, strings, identifiers, and keywords. Constants of most non-numeric types are first classified as strings. The SQL language definition allows specifying type names with constant strings. For example, the query:

```
openGauss=# SELECT text 'Origin' AS "label", point '(0,0)' AS "value";
 label  | value
--------+-------
 Origin | (0,0)
(1 row)
```

has two literal constants, of type **text** and **point**. If a type is not specified for a string literal, then the placeholder type **unknown** is assigned initially.

There are four fundamental SQL constructs requiring distinct type conversion rules in GaussDB parser:

- Function calls

  Much of the SQL type system is built around a rich set of functions. Functions can have one or more arguments. Since SQL permits function overloading, the function name alone does not uniquely identify the function to be called. The parser must select the right function based on the data types of the supplied arguments.
- Operators

  SQL allows expressions with prefix and postfix unary (one-argument) operators, as well as binary (two-argument) operators. Like functions, operators can be overloaded, so the same problem of selecting the right operator exists.
- Value storage

SQL **INSERT** and **UPDATE** statements place the results of expressions into a table. The expressions in the statement must be matched up with, and perhaps converted to, the types of the target columns.

- UNION, CASE, and Related Constructs

  Since all query results from a unionized **SELECT** statement must appear in a single set of columns, the types of the results of each **SELECT** clause must be matched up and converted to a uniform set. Similarly, the result expressions of a **CASE** construct must be converted to a common type so that the **CASE** expression as a whole has a known output type. The same holds for **ARRAY** constructs, and for the **GREATEST** and **LEAST** functions.

The system catalog pg_cast stores information about which conversions, or casts, exist between which data types, and how to perform those conversions. For details, see **PG_CAST**.

The return type and conversion behavior of an expression are determined during semantic analysis. Data types are divided into several basic type categories, including Boolean, **numeric**, **string**, **bitstring**, **datetime**, **timespan**, **geometric**, and **network**. Within each category there can be one or more preferred types, which are preferred when there is a choice of possible types. With careful selection of preferred types and available implicit casts, it is possible to ensure that ambiguous expressions (those with multiple candidate parsing solutions) can be resolved in a useful way.

All type conversion rules are designed based on the following principles:

- Implicit conversions should never have surprising or unpredictable outcomes.
- There should be no extra overhead in the parser or executor if a query does not need implicit type conversion. That is, if a query is well-formed and the types already match, then the query should execute without spending extra time in the parser and without introducing unnecessary implicit conversion calls in the query.
- Additionally, if a query usually requires an implicit conversion for a function, and if then the user defines a new function with the correct argument types, the parser should use this new function.

11.7.2 Operators

Operator Type Resolution

1. Select the operators to be considered from the **pg_operator** system catalog. Considered operators are those with the matching name and argument count. If the search path finds multiple available operators, only the most suitable one is considered.
2. Look for the best match.
   a. Discard candidate operators for which the input types do not match and cannot be converted (using an implicit conversion) to match. **unknown** literals are assumed to be convertible to anything for this purpose. If only one candidate remains, use it; else continue to the next step.
   b. Run through all candidates and keep those with the most exact matches on input types. Domains are considered the same as their base type for this purpose. Keep all candidates if none survives these tests. If only one candidate remains, use it; else continue to the next step.

c. Run through all candidates and keep those that accept preferred types (of the input data type's type category) at the most positions where type conversion will be required. Keep all candidates if none accepts preferred types. If only one candidate remains, use it; else continue to the next step.

d. If any input arguments are **unknown**, check the type categories accepted at those argument positions by the remaining candidates. At each position, select the string category if any candidate accepts that category. (This bias towards string is appropriate since an unknown-type literal looks like a string.) Otherwise, if all the remaining candidates accept the same type category, select that category; otherwise fail because the correct choice cannot be deduced without more clues. Now discard candidates that do not accept the selected type category. Furthermore, if any candidate accepts a preferred type in that category, discard candidates that accept non-preferred types for that argument. Keep all candidates if none survives these tests. If only one candidate remains, use it; else continue to the next step.

e. If there are both **unknown** and known-type arguments, and all the known-type arguments have the same type, assume that the **unknown** arguments are also of that type, and check which candidates can accept that type at the unknown-argument positions. If exactly one candidate passes this test, use it. Otherwise, an error occurs.

Examples

Example 1: Use factorial operator type resolution. There is only one factorial operator (postfix !) defined in the system catalog, and it takes an argument of type **bigint**. The scanner assigns an initial type of **bigint** to the argument in this query expression:

```
openGauss=# SELECT 40 ! AS "40 factorial";

                 40 factorial
--------------------------------------------------
 815915283247897734345611269596115894272000000000
(1 row)
```

So the parser does a type conversion on the operand and the query is equivalent to:

```
openGauss=# SELECT CAST(40 AS bigint) ! AS "40 factorial";
```

Example 2: String concatenation operator type resolution. A string-like syntax is used for working with string types and for working with complex extension types. Strings with unspecified type are matched with likely operator candidates. An example with one unspecified argument:

```
openGauss=# SELECT text 'abc' || 'def' AS "text and unknown";
 text and unknown
------------------
 abcdef
(1 row)
```

In this example, the parser looks for an operator whose parameters are of the text type. Such an operator is found.

Here is a concatenation of two values of unspecified types:

```
openGauss=# SELECT 'abc' || 'def' AS "unspecified";
 unspecified
```

```
-------------
 abcdef
(1 row)
```

NOTE

In this case there is no initial hint for which type to use, since no types are specified in the query. So, the parser looks for all candidate operators and finds that there are candidates accepting both string-category and bit-string-category inputs. Since string category is preferred when available, that category is selected, and then the preferred type for strings, **text**, is used as the specific type to resolve the unknown-type literals as.

Example 3: Absolute-value and negation operator type resolution. The GaussDB operator catalog has several entries for the prefix operator @. All the entries implement absolute-value operations for various numeric data types. One of these entries is for type **float8**, which is the preferred type in the numeric category. Therefore, GaussDB will use that entry when faced with an unknown input:

```
openGauss=# SELECT @ '-4.5' AS "abs";
 abs
-----
 4.5
(1 row)
```

Here the system has implicitly resolved the unknown-type literal as type **float8** before applying the chosen operator.

Example 4: Use the array inclusion operator type resolution as an example. Here is another example of resolving an operator with one known and one unknown input:

```
openGauss=# SELECT array[1,2] <@ '{1,2,3}' as "is subset";
 is subset
-----------
 t
(1 row)
```

NOTE

The GaussDB operator catalog has several entries for the infix operator <@, but the only two that could possibly accept an integer array on the left side are array inclusion (anyarray <@ anyarray) and range inclusion (anyelement <@ anyrange). Since none of these polymorphic pseudo-types (see **Pseudo-Types**) is considered preferred, the parser cannot resolve the ambiguity on that basis. However, the last resolution rule tells it to assume that the unknown-type literal is of the same type as the other input, that is, integer array. Now only one of the two operators can match, so array inclusion is selected. (Had range inclusion been selected, we would have gotten an error, because the string does not have the right format to be a range literal.)

11.7.3 Functions

Function Type Resolution

1. Select the functions to be considered from the **pg_proc** system catalog. If a non-schema-qualified function name was used, the functions in the current search path are considered. If a qualified function name was given, only functions in the specified schema are considered.

   If the search path finds multiple functions of different argument types, a proper function in the path is considered.

2. Check for a function accepting exactly the input argument types. If the function exists, use it. Cases involving **unknown** will never find a match at this step.
3. If no exact match is found, see if the function call appears to be a special type conversion request.
4. Look for the best match.
   a. Discard candidate functions for which the input types do not match and cannot be converted (using an implicit conversion) to match. **unknown** literals are assumed to be convertible to anything for this purpose. If only one candidate remains, use it; else continue to the next step.
   b. Run through all candidates and keep those with the most exact matches on input types. Domains are considered the same as their base type for this purpose. Keep all candidates if none has exact matches. If only one candidate remains, use it; else continue to the next step.
   c. Run through all candidates and keep those that accept preferred types at the most positions where type conversion will be required. Keep all candidates if none accepts preferred types. If only one candidate remains, use it; else continue to the next step.
   d. If any input arguments are **unknown**, check the type categories accepted at those argument positions by the remaining candidates. At each position, select the string category if any candidate accepts that category. (This bias towards string is appropriate since an unknown-type literal looks like a string.) Otherwise, if all the remaining candidates accept the same type category, select that category; otherwise fail because the correct choice cannot be deduced without more clues. Now discard candidates that do not accept the selected type category. Furthermore, if any candidate accepts a preferred type in that category, discard candidates that accept non-preferred types for that argument. Keep all candidates if none survives these tests. If only one candidate remains, use it; else continue to the next step.
   e. If there are both **unknown** and known-type arguments, and all the known-type arguments have the same type, assume that the **unknown** arguments are also of that type, and check which candidates can accept that type at the **unknown**-argument positions. If exactly one candidate passes this test, use it. Otherwise, an error occurs.

Examples

Example 1: Use the rounding function argument type resolution as the first example. There is only one **round** function that takes two arguments; it takes a first argument of type **numeric** and a second argument of type **integer**. So the following query automatically converts the first argument of type **integer** to **numeric**:

```
openGauss=# SELECT round(4, 4);
 round
--------
 4.0000
(1 row)
```

That query is actually transformed by the parser to:

```
openGauss=# SELECT round(CAST (4 AS numeric), 4);
```

Since numeric constants with decimal points are initially assigned the type **numeric**, the following query will require no type conversion and therefore might be slightly more efficient:

```
openGauss=# SELECT round(4.0, 4);
```

Example 2: Use the substring function type resolution as the second example. There are several **substr** functions, one of which takes types **text** and **integer**. If called with a string constant of unspecified type, the system chooses the candidate function that accepts an argument of the preferred category **string** (namely of type **text**).

```
openGauss=# SELECT substr('1234', 3);
 substr
--------
     34
(1 row)
```

If the string is declared to be of type **varchar**, as might be the case if it comes from a table, then the parser will try to convert it to become **text**:

```
openGauss=# SELECT substr(varchar '1234', 3);
 substr
--------
     34
(1 row)
```

This is transformed by the parser to effectively become:

```
openGauss=# SELECT substr(CAST (varchar '1234' AS text), 3);
```

NOTE

The parser learns from the **pg_cast** catalog that **text** and **varchar** are binary-compatible, meaning that one can be passed to a function that accepts the other without doing any physical conversion. Therefore, no type conversion is really inserted in this case.

And, if the function is called with an argument of type **integer**, the parser will try to convert that to **text**:

```
openGauss=# SELECT substr(1234, 3);
substr
--------
 34
(1 row)
```

This is transformed by the parser to effectively become:

```
openGauss=# SELECT substr(CAST (1234 AS text), 3);
 substr
--------
     34
(1 row)
```

11.7.4 Value Storage

Value Storage Type Resolution

1. Search for an exact match with the target column.
2. Try to convert the expression to the target type. This will succeed if there is a registered cast between the two types. If the expression is an unknown-type literal, the content of the literal string will be fed to the input conversion routine for the target type.

3. Check to see if there is a sizing cast for the target type. A sizing cast is a cast from that type to itself. If one is found in the **pg_cast** catalog, apply it to the expression before storing into the destination column. The implementation function for such a cast always takes an extra parameter of type **integer**. The parameter receives the destination column's **atttypmod** value (typically its declared length, although the interpretation of **atttypmod** varies for different data types), and may take a third Boolean parameter that says whether the cast is explicit or implicit. The cast function is responsible for applying any length-dependent semantics such as size checking or truncation.

Examples

Use the **character** storage type conversion as an example. For a target column declared as **character(20)** the following statement shows that the stored value is sized correctly:

```
openGauss=# CREATE TABLE tpcds.value_storage_t1 (
    VS_COL1 CHARACTER(20)
);
openGauss=# INSERT INTO tpcds.value_storage_t1 VALUES('abcdef');
openGauss=# SELECT VS_COL1, octet_length(VS_COL1) FROM tpcds.value_storage_t1;
       vs_col1        | octet_length
----------------------+--------------
 abcdef               |           20
(1 row)
)

openGauss=# DROP TABLE tpcds.value_storage_t1;
```

NOTE

What has happened here is that the two unknown literals are resolved to **text** by default, allowing the **||** operator to be resolved as **text** concatenation. Then the **text** result of the operator is converted to **bpchar** ("blank-padded char", the internal name of the **character** data type) to match the target column type. Since the conversion from **text** to **bpchar** is binary-coercible, this conversion does not insert any real function call. Finally, the sizing function **bpchar(bpchar, integer, Boolean)** is found in the system catalog and used for the operator's result and the stored column length. This type-specific function performs the required length check and addition of padding spaces.

11.7.5 UNION, CASE, and Related Constructs

SQL **UNION** constructs must match up possibly dissimilar types to become a single result set. The resolution algorithm is applied separately to each output column of a union query. The **INTERSECT** and **EXCEPT** construct resolve dissimilar types in the same way as **UNION**. The **CASE**, **ARRAY**, **VALUES**, **GREATEST** and **LEAST** constructs use the identical algorithm to match up their component expressions and select a result data type.

Type Resolution for UNION, CASE, and Related Constructs

- If all inputs are of the same type, and it is not **unknown**, resolve as that type.
- If all inputs are of type **unknown**, resolve as type **text** (the preferred type of the string category). Otherwise, **unknown** inputs are ignored.
- If the inputs are not all of the same type category, a failure will be resulted. (Type **unknown** is not included in this case.)

- If the inputs are all of the same type category, choose the top preferred type in that category. (Exception: The UNION operation regards the type of the first branch as the selected type.)

  NOTE

  typcategory in the **pg_type** system catalog indicates the data type category. **typispreferred** indicates whether a type is preferred in **typcategory**.

- Convert all inputs to the selected type. (Retain the original lengths of strings). Fail if there is not an implicit conversion from a given input to the selected type.
- If the input contains the json, txid_snapshot, sys_refcursor, or geometry type, **UNION** cannot be performed.

Type Resolution for CASE and COALESCE in TD Compatibility Type

- If all inputs are of the same type, and it is not **unknown**, resolve as that type.
- If all inputs are of type **unknown**, resolve as type **text**.
- If inputs are of the string type (including **unknown** which is resolved as type **text**) and digit type, resolve as the string type. If the inputs are not of the two types, an error will be reported.
- If the inputs are all of the same type category, choose the top preferred type in that category.
- Convert all inputs to the selected type. Fail if there is not an implicit conversion from a given input to the selected type.

Type Resolution for CASE in ORA Compatibility Type

decode(expr, search1, result1, search2, result2, ..., defresult): When the **sql_beta_feature** is set to **a_style_coerce**, the final return value type of the expression is set to the data type of result1 or a higher-precision data type in the same type as result1. (For example, numeric and int are both numeric data types, but numeric has higher precision and priority than int.) For CASE WHEN, the behavior is the same as the default behavior in ORA-compatible mode.

- If all inputs are of the same type, and it is not **unknown**, resolve as that type. Otherwise, proceed to the next step.
- Set the data type of result1 to the final return value type preferType, which belongs to preferCategory.
- Consider the data types of result2, result3, and defresult in sequence. If the type category is also preferCategory, which is the same as that of result1, check whether the precision (priority) is higher than that of preferType. If it is, update preferType to a data type with a higher precision. If the type category is not preferCategory, check whether the category can be implicitly converted to preferType. If it cannot, an error is reported.
- Uses the data type recorded by preferType as the return value type of the expression. The expression result is implicitly converted to this data type.

Note:

There is a special case where the character type of a super-large number is converted to the numeric type, for example, **select decode(1, 2, 2, '53465465676465454657567678676')**, in which the large number exceeds the range of the bigint and double types. If result1 is of the numeric type and does not meet the condition that all inputs are of the same type, the type of the return value is set to numeric to be compatible with this special case.

Note 2:

Priority of the numeric types: numeric > float8 > float4 > int8 > int4 > int2 > int1

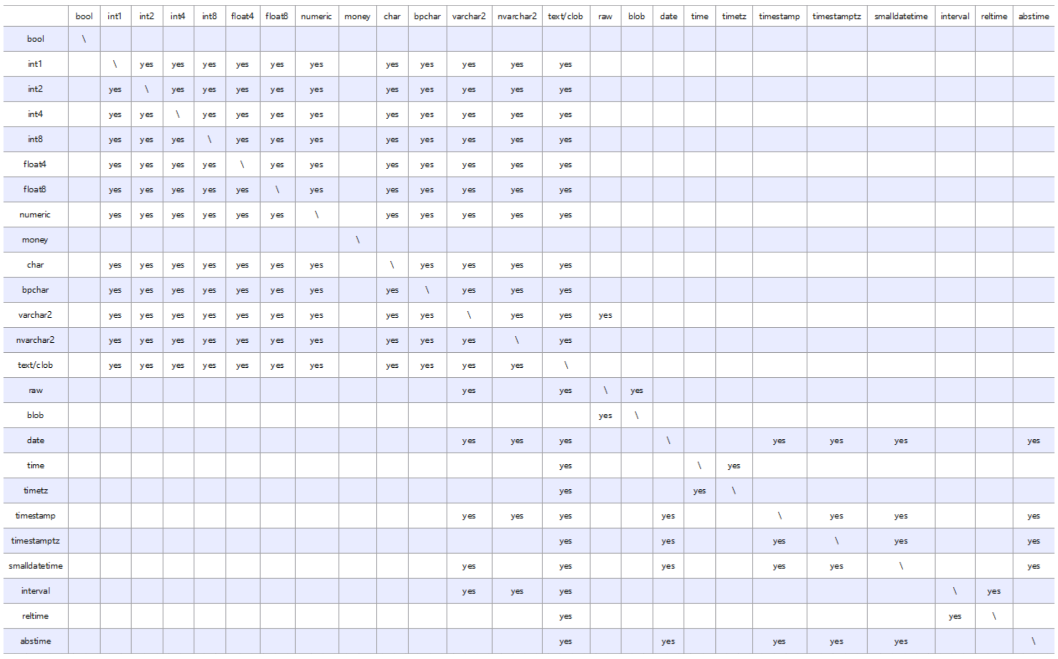
Priority of the character types: text > varchar = nvarchar2 > bpchar > char

Priority of date types: timestamptz > timestamp > smalldatetime > date > abstime > timetz > time

Priority of date span types: interval > tinterval > reltime.

Note 3:

The following figure shows the supported implicit type conversion when **set sql_beta_feature** is set to **'a_style_coerce'** in ORA compatibility mode. \ indicates that conversion is not required, **yes** indicates that conversion is supported, and the blank value indicates that conversion is not supported.



	bool	int1	int2	int4	int8	float4	float8	numeric	money	char	bpchar	varchar2	nvarchar2	text/clob	raw	blob	date	time	timetz	timestamp	timestamptz	smalldatetime	interval	reltime	abstime
bool	\																								
int1		\	yes	yes	yes	yes	yes	yes		yes	yes	yes	yes	yes											
int2		yes	\	yes	yes	yes	yes	yes		yes	yes	yes	yes	yes											
int4		yes	yes	\	yes	yes	yes	yes		yes	yes	yes	yes	yes											
int8		yes	yes	yes	\	yes	yes	yes		yes	yes	yes	yes	yes											
float4		yes	yes	yes	yes	\	yes	yes		yes	yes	yes	yes	yes											
float8		yes	yes	yes	yes	yes	\	yes		yes	yes	yes	yes	yes											
numeric		yes	yes	yes	yes	yes	yes	\		yes	yes	yes	yes	yes											
money									\																
char		yes	yes	yes	yes	yes	yes	yes		\	yes	yes	yes	yes											
bpchar		yes	yes	yes	yes	yes	yes	yes		yes	\	yes	yes	yes											
varchar2		yes	yes	yes	yes	yes	yes	yes		yes	yes	\	yes	yes	yes										
nvarchar2		yes	yes	yes	yes	yes	yes	yes		yes	yes	yes	\	yes											
text/clob		yes	yes	yes	yes	yes	yes	yes		yes	yes	yes	yes	\											
raw												yes		yes	\	yes									
blob															yes	\									
date												yes	yes	yes			\			yes	yes	yes			yes
time														yes				\	yes						
timetz														yes				yes	\						
timestamp												yes	yes	yes			yes			\	yes	yes			yes
timestamptz														yes			yes			yes	\	yes			yes
smalldatetime												yes		yes			yes			yes	yes	\			yes
interval												yes	yes	yes									\	yes	
reltime														yes									yes	\	
abstime														yes			yes			yes	yes	yes			\

Examples

Example 1: Use type resolution with underspecified types in a union as the first example. Here, the unknown-type literal **'b'** will be resolved to type **text**.

```
openGauss=# SELECT text 'a' AS "text" UNION SELECT 'b';
 text
------
 a
 b
(2 rows)
```

Example 2: Use type resolution in a simple union as the second example. The literal **1.2** is of type **numeric**, and the **integer** value **1** can be cast implicitly to **numeric**, so that type is used.

```
openGauss=# SELECT 1.2 AS "numeric" UNION SELECT 1;
 numeric
---------
       1
     1.2
(2 rows)
```

Example 3: Use type resolution in a transposed union as the third example. Since type **real** cannot be implicitly cast to **integer**, but **integer** can be implicitly cast to **real**, the union result type is resolved as **real**.

```
openGauss=# SELECT 1 AS "real" UNION SELECT CAST('2.2' AS REAL);
 real
------
    1
  2.2
(2 rows)
```

Example 4: In the **TD** type, if input parameters for **COALESCE** are of **int** and **varchar** types, resolve as type **varchar**. In the **A** type, an error is reported.

```
-- In the A type, create the a_1 database compatible with A.
openGauss=# CREATE DATABASE a_1 dbcompatibility = 'A';

-- Switch to the a_1 database.
openGauss=# \c a_1

-- Create the t1 table.
a_1=# CREATE TABLE t1(a int, b varchar(10));

-- Show the execution plan of a statement for querying the types int and varchar of input parameters for
COALESCE.
a_1=# EXPLAIN SELECT coalesce(a, b) FROM t1;
ERROR:  COALESCE types integer and character varying cannot be matched
LINE 1: EXPLAIN SELECT coalesce(a, b) FROM t1;
                                 ^
CONTEXT:  referenced column: coalesce

-- Delete the table.
a_1=# DROP TABLE t1;

-- Switch to openGauss.
a_1=# \c openGauss

-- In the TD type, create the td_1 database compatible with Teradata.
openGauss=# CREATE DATABASE td_1 dbcompatibility = 'C';

-- Switch to the td_1 database.
openGauss=# \c td_1

-- Create the t2 table.
td_1=# CREATE TABLE t2(a int, b varchar(10));

-- Show the execution plan of a statement for querying the types int and varchar of input parameters for
COALESCE.
td_1=# EXPLAIN VERBOSE select coalesce(a, b) from t2;
                                   QUERY PLAN
---------------------------------------------------------------------------------------
 Data Node Scan  (cost=0.00..0.00 rows=0 width=0)
   Output: (COALESCE((t2.a)::character varying, t2.b))
   Node/s: All dbnodes
   Remote query: SELECT COALESCE(a::character varying, b) AS "coalesce" FROM public.t2
(4 rows)

-- Delete the table.
td_1=# DROP TABLE t2;

-- Switch to openGauss.
td_1=# \c openGauss
```

```
-- Delete databases in A and TD types.
openGauss=# DROP DATABASE a_1;
openGauss=# DROP DATABASE td_1;
```

Example 5: In ORA mode, set the final return value type of the expression to the data type of result1 or a higher-precision data type whose category is the same as that of the data type of result1.

```
-- In the ORA type, create the ora_1 database compatible with ORA.
openGauss=# CREATE DATABASE ora_1 dbcompatibility = 'A';

-- Switch to the ora_1 database.
openGauss=# \c ora_1

-- Enable the decode compatibility parameters.
set sql_beta_feature='a_style_coerce';

-- Create the t1 table.
ora_1=# CREATE TABLE t1(c_int int, c_float8 float8, c_char char(10), c_text text, c_date date);

-- Insert data.
ora_1=# INSERT INTO t1 VALUES(1, 2, '3', '4', date '12-10-2010');

-- The data type of result1 is char and that of defresult is text. The precision of text is higher, and the type
of the return value is changed to text from char.
ora_1=# SELECT decode(1, 2, c_char, c_text) AS result, pg_typeof(result) FROM t1;
 result | pg_typeof
--------+-----------
 4      | text
(1 row)

-- The data type of result1 is int, which is a numeric type. The type of the return value is set to numeric.
ora_1=# SELECT decode(1, 2, c_int, c_float8) AS result, pg_typeof(result) FROM t1;
 result | pg_typeof
--------+-----------
      2 | numeric
(1 row)

-- The implicit conversion from the data type of defresult to that of result1 does not exist. An error is
reported.
ora_1=# SELECT decode(1, 2, c_int, c_date) FROM t1;
ERROR:  CASE types integer and timestamp without time zone cannot be matched
LINE 1: SELECT decode(1, 2, c_int, c_date) FROM t1;
                                   ^
CONTEXT:  referenced column: c_date

-- Disable the decode compatibility parameters.
set sql_beta_feature='none';

-- Delete the table.
ora_1=# DROP TABLE t1;
DROP TABLE

-- Switch to the postgres database:
ora_1=# \c postgres

-- Delete the database in ORA mode.
openGauss=# DROP DATABASE ora_1;
DROP DATABASE
```

11.8 Full Text Search

11.8.1 Introduction

11.8.1.1 Full-Text Retrieval

Textual search operators have been used in databases for years. GaussDB has **~**, **~***, **LIKE**, and **ILIKE** operators for textual data types, but they lack many essential properties required by modern information systems. They can be supplemented by indexes and dictionaries.

Text search lacks the following essential properties required by information systems:

- There is no linguistic support, even for English.

  Regular expressions are not sufficient because they cannot easily handle derived words. For example, you might miss documents that contain **satisfies**, although you probably would like to find them when searching for **satisfy**. It is possible to use **OR** to search for multiple derived forms, but this is tedious and error-prone, because some words can have several thousand derivatives.
- They provide no ordering (ranking) of search results, which makes them ineffective when thousands of matching documents are found.
- They tend to be slow because there is no index support, so they must process all documents for every search.

Full text indexing allows documents to be preprocessed and an index is saved for later rapid searching. Preprocessing includes:

- Parsing documents into tokens

  It is useful to identify various classes of tokens, for example, numbers, words, complex words, and email addresses, so that they can be processed differently. In principle, token classes depend on the specific application, but for most purposes it is adequate to use a predefined set of classes.
- Converting tokens into lexemes

  A lexeme is a string, just like a token, but it has been normalized so that different forms of the same word are made alike. For example, normalization almost always includes folding upper-case letters to lower-case, and often involves removal of suffixes (such as **s** or **es** in English). This allows searches to find variant forms of the same word, without tediously entering all the possible variants. Also, this step typically eliminates stop words, which are words that are so common that they are useless for searching. (In short, tokens are raw fragments of the document text, while lexemes are words that are believed useful for indexing and searching.) GaussDB uses dictionaries to perform this step and provides various standard dictionaries.
- Storing preprocessed documents optimized for searching

  For example, each document can be represented as a sorted array of normalized lexemes. Along with the lexemes, it is often desirable to store positional information for proximity ranking. Therefore, a document that contains a more "dense" region of query words is assigned with a higher rank than the one with scattered query words.

Dictionaries allow fine-grained control over how tokens are normalized. With appropriate dictionaries, you can define stop words that should not be indexed.

A data type **tsvector** is provided for storing preprocessed documents, along with a type **tsquery** for storing query conditions. For details, see **Text Search Types**. For details about the functions and operators available for these data types, see **Text Search Functions and Operators**. The match operator **@@**, which is the most important among those functions and operators, is introduced in **Basic Text Matching**.

11.8.1.2 What Is a Document?

A document is the unit of searching in a full text search system; for example, a magazine article or email message. The text search engine must be able to parse documents and store associations of lexemes (keywords) with their parent document. Later, these associations are used to search for documents that contain query words.

For searches within GaussDB, a document is normally a textual column within a row of a database table, or possibly a combination (concatenation) of such columns, perhaps stored in several tables or obtained dynamically. In other words, a document can be constructed from different parts for indexing and it might not be stored anywhere as a whole. Example:

```
openGauss=# SELECT d_dow || '-' || d_dom || '-' || d_fy_week_seq  AS identify_serials FROM tpcds.date_dim
WHERE d_fy_week_seq = 1;
identify_serials
------------------
 5-6-1
 0-8-1
 2-3-1
 3-4-1
 4-5-1
 1-2-1
 6-7-1
(7 rows)
```

NOTICE

Actually, in these example queries, **coalesce** should be used to prevent a single **NULL** attribute from causing a **NULL** result for the whole document.

Another possibility is to store the documents as simple text files in the file system. In this case, the database can be used to store the full text index and to execute searches, and some unique identifier can be used to retrieve the document from the file system. However, retrieving files from outside the database requires system administrator permissions or special function support, so this is usually less convenient than keeping all the data inside the database. Also, keeping everything inside the database allows easy access to document metadata to assist in indexing and display.

For text search purposes, each document must be reduced to the preprocessed **tsvector** format. Searching and relevance-based ranking are performed entirely on the **tsvector** representation of a document. The original text is retrieved only when the document has been selected for display to a user. We therefore often speak of the **tsvector** as being the document, but it is only a compact representation of the full document.

11.8.1.3 Basic Text Matching

Full text search in GaussDB is based on the match operator **@@**, which returns **true** if a **tsvector** (document) matches a **tsquery** (query). It does not matter which data type is written first:

```
openGauss=# SELECT 'a fat cat sat on a mat and ate a fat rat'::tsvector @@ 'cat & rat'::tsquery AS RESULT;
 result
----------
 t
(1 row)
openGauss=# SELECT 'fat & cow'::tsquery @@ 'a fat cat sat on a mat and ate a fat rat'::tsvector AS RESULT;
 result
----------
 f
(1 row)
```

As the above example suggests, a **tsquery** is not raw text, any more than a **tsvector** is. A tsquery contains search terms, which must be already-normalized lexemes, and may combine multiple terms using **AND**, **OR**, and **NOT** operators. For details, see **Text Search Types**. There are functions **to_tsquery** and **plainto_tsquery** that are helpful in converting user-written text into a proper tsquery, for example by normalizing words appearing in the text. Similarly, **to_tsvector** is used to parse and normalize a document string. So in practice a text search match would look more like this:

```
openGauss=# SELECT to_tsvector('fat cats ate fat rats') @@ to_tsquery('fat & rat') AS RESULT;
result
----------
 t
(1 row)
```

Observe that this match would not succeed if written as follows:

```
openGauss=# SELECT 'fat cats ate fat rats'::tsvector @@ to_tsquery('fat & rat')AS RESULT;
result
----------
 f
(1 row)
```

In the preceding match, no normalization of the word **rats** will occur. Therefore, **rats** does not match **rat**.

The **@@** operator also supports text input, allowing explicit conversion of a text string to **tsvector** or **tsquery** to be skipped in simple cases. The variants available are:

```
tsvector @@ tsquery
tsquery  @@ tsvector
text @@ tsquery
text @@ text
```

We already saw the first two of these. The form **text @@ tsquery** is equivalent to **to_tsvector(text) @@ tsquery**. The form **text @@ text** is equivalent to **to_tsvector(text) @@ plainto_tsquery(text)**.

11.8.1.4 Configurations

Full text search functionality includes the ability to do many more things: skip indexing certain words (stop words), process synonyms, and use sophisticated parsing, for example, parse based on more than just white space. This functionality is controlled by text search configurations. GaussDB comes with predefined configurations for many languages, and you can easily create your own configurations. (The **\dF** command of **gsql** shows all available configurations.)

During installation an appropriate configuration is selected and **default_text_search_config** is set accordingly in **postgresql.conf**. If you are using the same text search configuration for the entire database, you can use the value in **postgresql.conf**. To use different configurations throughout the database but the same configuration within any one database, use **ALTER DATABASE ... SET**. Otherwise, you can set **default_text_search_config** in each session.

Each text search function that depends on a configuration has an optional argument, so that the configuration to use can be specified explicitly. **default_text_search_config** is used only when this argument is omitted.

To make it easier to build custom text search configurations, a configuration is built up from simpler database objects. GaussDB's text search facility provides the following types of configuration-related database objects:

- Text search parsers break documents into tokens and classify each token (for example, as words or numbers).
- Text search dictionaries convert tokens to normalized form and reject stop words.
- Text search templates provide the functions underlying dictionaries. (A dictionary simply specifies a template and a set of parameters for the template.)
- Text search configurations select a parser and a set of dictionaries to use to normalize the tokens produced by the parser.

11.8.2 Tables and Indexes

11.8.2.1 Searching a Table

It is possible to do a full text search without an index.

- A simple query to print each row that contains the word **america** in its **body** column is as follows:

```
openGauss=# DROP SCHEMA IF EXISTS tsearch CASCADE;

openGauss=# CREATE SCHEMA tsearch;

openGauss=# CREATE TABLE tsearch.pgweb(id int, body text, title text, last_mod_date date);

openGauss=# INSERT INTO tsearch.pgweb VALUES(1, 'China, officially the People''s Republic of China
(PRC), located in Asia, is the world''s most populous state.', 'China', '2010-1-1');

openGauss=# INSERT INTO tsearch.pgweb VALUES(2, 'America is a rock band, formed in England in
1970 by multi-instrumentalists Dewey Bunnell, Dan Peek, and Gerry Beckley.', 'America', '2010-1-1');

openGauss=# INSERT INTO tsearch.pgweb VALUES(3, 'England is a country that is part of the United
Kingdom. It shares land borders with Scotland to the north and Wales to the west.', 'England',
'2010-1-1');

openGauss=# INSERT INTO tsearch.pgweb VALUES(4, 'Australia, officially the Commonwealth of
Australia, is a country comprising the mainland of the Australian continent, the island of Tasmania,
and numerous smaller islands.', 'Australia', '2010-1-1');

openGauss=# INSERT INTO tsearch.pgweb VALUES(6, 'Japan is an island country in East Asia.', 'Japan',
'2010-1-1');

openGauss=# INSERT INTO tsearch.pgweb VALUES(7, 'Germany, officially the Federal Republic of
Germany, is a sovereign state and federal parliamentary republic in central-western Europe.',
'Germany', '2010-1-1');

openGauss=# INSERT INTO tsearch.pgweb VALUES(8, 'France, is a sovereign state comprising territory
in western Europe and several overseas regions and territories.', 'France', '2010-1-1');
```

```
openGauss=# INSERT INTO tsearch.pgweb VALUES(9, 'Italy officially the Italian Republic, is a unitary
parliamentary republic in Europe.', 'Italy', '2010-1-1');

openGauss=# INSERT INTO tsearch.pgweb VALUES(10, 'India, officially the Republic of India, is a
country in South Asia.', 'India', '2010-1-1');

openGauss=# INSERT INTO tsearch.pgweb VALUES(11, 'Brazil, officially the Federative Republic of
Brazil, is the largest country in both South America and Latin America.', 'Brazil', '2010-1-1');

openGauss=# INSERT INTO tsearch.pgweb VALUES(12, 'Canada is a country in the northern half of
North America.', 'Canada', '2010-1-1');

openGauss=# INSERT INTO tsearch.pgweb VALUES(13, 'Mexico, officially the United Mexican States, is
a federal republic in the southern part of North America.', 'Mexico', '2010-1-1');

openGauss=# SELECT id, body, title FROM tsearch.pgweb WHERE to_tsvector('english', body) @@
to_tsquery('english', 'america');
 id |                                              body                                              |  title
----
+------------------------------------------------------------------------------------------------------------------
----+---------
  2 | America is a rock band, formed in England in 1970 by multi-instrumentalists Dewey Bunnell, Dan
Peek, and Gerry Beckley. | America
 11 | Brazil, officially the Federative Republic of Brazil, is the largest country in both South America
and Latin America.   | Brazil
 12 | Canada is a country in the northern half of North
America.                                                          | Canada
 13 | Mexico, officially the United Mexican States, is a federal republic in the southern part of North
America.              | Mexico
(4 rows)
```

This will also find related words, such as **America**, since all these are reduced to the same normalized lexeme.

The query above specifies that the **english** configuration is to be used to parse and normalize the strings. Alternatively we could omit the configuration parameters, and use the configuration set by **default_text_search_config**.

```
openGauss=# SHOW default_text_search_config;
 default_text_search_config
----------------------------
 pg_catalog.english
(1 row)

openGauss=# SELECT id, body, title FROM tsearch.pgweb WHERE to_tsvector(body) @@
to_tsquery('america');
 id |                                              body                                              |  title
----
+------------------------------------------------------------------------------------------------------------------
----+---------
  2 | America is a rock band, formed in England in 1970 by multi-instrumentalists Dewey Bunnell, Dan
Peek, and Gerry Beckley. | America
 11 | Brazil, officially the Federative Republic of Brazil, is the largest country in both South America
and Latin America.   | Brazil
 12 | Canada is a country in the northern half of North
America.                                                          | Canada
 13 | Mexico, officially the United Mexican States, is a federal republic in the southern part of North
America.              | Mexico
(4 rows)
```

- A more complex example to select the ten most recent documents that contain **north** and **america** in the **title** or **body** column is as follows:

```
openGauss=# SELECT title FROM tsearch.pgweb WHERE to_tsvector(title || ' ' || body) @@
to_tsquery('north & america') ORDER BY last_mod_date DESC LIMIT 10;
 title
--------
 Canada
 Mexico
(2 rows)
```

For clarity we omitted the **coalesce** function calls which would be needed to find rows that contain **NULL** in one of the two columns.

The preceding examples show queries without using indexes. Most applications will find this approach too slow. Therefore, practical use of text searching usually requires creating an index, except perhaps for occasional ad-hoc searches.

11.8.2.2 Creating an Index

You can create a **GIN** index to speed up text searches:

```
openGauss=# CREATE INDEX pgweb_idx_1 ON tsearch.pgweb USING gin(to_tsvector('english', body));
```

The **to_tsvector** function comes in to two versions: the 1-argument version and the 2-argument version. When the 1-argument version is used, the system uses the configuration specified by **default_text_search_config** by default.

Notice that the 2-argument version of **to_tsvector** is used for index creation. Only text search functions that specify a configuration name can be used in expression indexes. This is because the index contents must be unaffected by **default_text_search_config**, whose value can be changed at any time. If they were affected, the index contents might be inconsistent, because different entries could contain **tsvectors** that were created with different text search configurations, and there would be no way to guess which was which. It would be impossible to dump and restore such an index correctly.

Because the two-argument version of **to_tsvector** was used in the index above, only a query reference that uses the 2-argument version of **to_tsvector** with the same configuration name will use that index. That is, **WHERE to_tsvector('english', body) @@ 'a & b'** can use the index, but **WHERE to_tsvector(body) @@ 'a & b'** cannot. This ensures that an index will be used only with the same configuration used to create the index entries.

It is possible to set up more complex expression indexes wherein the configuration name is specified by another column. Example:

```
openGauss=# CREATE INDEX pgweb_idx_2 ON tsearch.pgweb USING gin(to_tsvector('ngram', body));
```

where **body** is a column in the **pgweb** table. This allows mixed configurations in the same index while recording which configuration was used for each index entry. This would be useful, for example, if the document collection contained documents in different languages. Again, queries that are meant to use the index must be phrased to match, for example, **WHERE to_tsvector(config_name, body) @@ 'a & b'** must match **to_tsvector** in the index.

Indexes can even concatenate columns:

```
openGauss=# CREATE INDEX pgweb_idx_3 ON tsearch.pgweb USING gin(to_tsvector('english', title || ' ' || 
body));
```

Another approach is to create a separate **tsvector** column to hold the output of **to_tsvector**. This example is a concatenation of **title** and **body**, using **coalesce** to ensure that one column will still be indexed when the other is **NULL**:

```
openGauss=# ALTER TABLE tsearch.pgweb ADD COLUMN textsearchable_index_col tsvector;
openGauss=# UPDATE tsearch.pgweb SET textsearchable_index_col = to_tsvector('english', coalesce(title,'') 
|| ' ' || coalesce(body,''));
```

Then, create a GIN index to speed up the search:

```
openGauss=# CREATE INDEX textsearch_idx_4 ON tsearch.pgweb USING gin(textsearchable_index_col);
```

Now you are ready to perform a fast full text search:

```
openGauss=# SELECT title
FROM tsearch.pgweb
WHERE textsearchable_index_col @@ to_tsquery('north & america')
ORDER BY last_mod_date DESC
LIMIT 10;

 title
--------
 Canada
 Mexico
(2 rows)
```

One advantage of the separate-column approach over an expression index is that it is unnecessary to explicitly specify the text search configuration in queries in order to use the index. As shown in the preceding example, the query can depend on **default_text_search_config**. Another advantage is that searches will be faster, since it will not be necessary to redo the **to_tsvector** calls to verify index matches. The expression-index approach is simpler to set up, however, and it requires less disk space since the **tsvector** representation is not stored explicitly.

11.8.2.3 Constraints on Index Use

The following is an example of index use:

```
openGauss=# create table table1 (c_int int,c_bigint bigint,c_varchar varchar,c_text text)
with(orientation=row);

openGauss=# create text search configuration ts_conf_1(parser=POUND);
openGauss=# create text search configuration ts_conf_2(parser=POUND) with(split_flag='%');

openGauss=# set default_text_search_config='ts_conf_1';
openGauss=# create index idx1 on table1 using gin(to_tsvector(c_text));

openGauss=# set default_text_search_config='ts_conf_2';
openGauss=# create index idx2 on table1 using gin(to_tsvector(c_text));

openGauss=# select c_varchar,to_tsvector(c_varchar) from table1 where to_tsvector(c_text) @@
plainto_tsquery('¥#@...&**')  and to_tsvector(c_text) @@ plainto_tsquery('Company ')  and c_varchar is not
null order by 1 desc limit 3;
```

In this example, **table1** has two GIN indexes created on the same column **c_text**, **idx1** and **idx2**, but these two indexes are created under different settings of **default_text_search_config**. Differences between this example and the scenario where one table has common indexes created on the same column are as follows:

- GIN indexes use different parsers (that is, different delimiters). In this case, the index data of **idx1** is different from that of **idx2**.
- In the specified scenario, the index data of multiple common indexes created on the same column is the same.

As a result, using **idx1** and **idx2** for the same query returns different results.

Constraints

Concluding the example above, when:

- Multiple GIN indexes are created on the same column of the same table.

- The GIN indexes use different parsers (that is, different delimiters).
- The column is used in a query, and an index scan is used in the execution plan.

  To avoid different query results caused by different GIN indexes, ensure that only one GIN index is available on a column of the physical table.

11.8.3 Controlling Text Search

11.8.3.1 Parsing Documents

GaussDB provides function **to_tsvector** for converting a document to the **tsvector** data type.

```
to_tsvector([ config regconfig, ] document text) returns tsvector
```

to_tsvector parses a textual document into tokens, reduces the tokens to lexemes, and returns a **tsvector**, which lists the lexemes together with their positions in the document. The document is processed according to the specified or default text search configuration. Here is a simple example:

```
openGauss=# SELECT to_tsvector('english', 'a fat  cat sat on a mat - it ate a fat rats');
                  to_tsvector
-----------------------------------------------------
 'ate':9 'cat':3 'fat':2,11 'mat':7 'rat':12 'sat':4
```

In the preceding example we see that the resulting **tsvector** does not contain the words **a**, **on**, or **it**, the word **rats** became **rat**, and the punctuation sign (-) was ignored.

The **to_tsvector** function internally calls a parser which breaks the document text into tokens and assigns a type to each token. For each token, a list of dictionaries is consulted. The list varies with the token type. The first dictionary that recognizes the token emits one or more normalized lexemes to represent the token. Example:

- **rats** became **rat** because one of the dictionaries recognized that the word **rats** is a plural form of **rat**.
- Some words are recognized as stop words (see **Stop Words**), which causes them to be ignored since they occur too frequently to be useful in searching. In our example these are **a**, **on**, and **it**.
- If no dictionary in the list recognizes the token then it is also ignored. In this example that happened to the punctuation sign (-) because there are no dictionaries assigned for its token type (**Space symbols**), meaning space tokens will never be indexed.

The choices of parser, dictionaries and which types of tokens to index are determined by the selected text search configuration. It is possible to have many different configurations in the same database, and predefined configurations are available for various languages. In our example we used the default configuration **english** for the English language.

The function **setweight** can be used to label the entries of a **tsvector** with a given weight, where a weight is one of the letters **A**, **B**, **C**, or **D**. This is typically used to mark entries coming from different parts of a document, such as title versus body. Later, this information can be used for ranking of search results.

Because **to_tsvector(NULL)** will return **NULL**, you are advised to use **coalesce** whenever a column might be null. Here is the recommended method for creating a **tsvector** from a structured document:

```
openGauss=# CREATE TABLE tsearch.tt (id int, title text, keyword text, abstract text, body text, ti tsvector);

openGauss=# INSERT INTO tsearch.tt(id, title, keyword, abstract, body) VALUES (1, 'China', 'Beijing', 'China','China, officially the People''s Republic of China (PRC), located in Asia, is the world''s most populous state.');

openGauss=# UPDATE tsearch.tt SET ti =
    setweight(to_tsvector(coalesce(title,'')), 'A')    ||
    setweight(to_tsvector(coalesce(keyword,'')), 'B')  ||
    setweight(to_tsvector(coalesce(abstract,'')), 'C') ||
    setweight(to_tsvector(coalesce(body,'')), 'D');
openGauss=# DROP TABLE tsearch.tt;
```

In this example, **setweight** is used to label the source of each lexeme in the finished **tsvector**, and then the labeled **tsvector** values are merged using the tsvector concatenation operator **||**. For details about these operations, see **Manipulating tsvector**.

11.8.3.2 Parsing Queries

GaussDB provides functions **to_tsquery** and **plainto_tsquery** for converting a query to the **tsquery** data type. **to_tsquery** offers access to more features than **plainto_tsquery**, but is less forgiving about its input.

```
to_tsquery([ config regconfig, ] querytext text) returns tsquery
```

to_tsquery creates a **tsquery** value from **querytext**, which must consist of single tokens separated by the Boolean operators **&** (AND), **|** (OR), and **!** (NOT). These operators can be grouped using parentheses. In other words, the input to **to_tsquery** must follow the general rules for **tsquery** input, as described in **Text Search Types**. The difference is that while basic **tsquery** input takes the tokens at face value, **to_tsquery** normalizes each token to a lexeme using the specified or default configuration, and discards any tokens that are stop words according to the configuration. Example:

```
openGauss=# SELECT to_tsquery('english', 'The & Fat & Rats');
   to_tsquery
---------------
 'fat' & 'rat'
(1 row)
```

As in basic **tsquery** input, **weight(s)** can be attached to each lexeme to restrict it to match only **tsvector** lexemes of those **weight(s)**. Example:

```
openGauss=# SELECT to_tsquery('english', 'Fat | Rats:AB');
    to_tsquery
------------------
 'fat' | 'rat':AB
(1 row)
```

Also, the asterisk (*) can be attached to a lexeme to specify prefix matching:

```
openGauss=# SELECT to_tsquery('supern:*A & star:A*B');
        to_tsquery
--------------------------
 'supern':*A & 'star':*AB
(1 row)
```

Such a lexeme will match any word having the specified string and weight in a **tsquery**.

```
plainto_tsquery([ config regconfig, ] querytext text) returns tsquery
```

plainto_tsquery transforms unformatted text **querytext** to **tsquery**. The text is parsed and normalized much as for **to_tsvector**, and then the **&** (AND) Boolean operator is inserted between surviving words.

Example:

```
openGauss=# SELECT plainto_tsquery('english', 'The Fat Rats');
 plainto_tsquery
-----------------
 'fat' & 'rat'
(1 row)
```

Note that **plainto_tsquery** cannot recognize Boolean operators, weight labels, or prefix-match labels in its input:

```
openGauss=# SELECT plainto_tsquery('english', 'The Fat & Rats:C');
   plainto_tsquery
---------------------
 'fat' & 'rat' & 'c'
(1 row)
```

Here, all the input punctuation was discarded as being space symbols.

11.8.3.3 Ranking Search Results

Ranking attempts to measure how relevant documents are to a particular query, so that when there are many matches the most relevant ones can be shown first. GaussDB provides two predefined ranking functions. The functions take into account lexical, proximity, and structural information; that is, they consider how often the query terms appear in the document, how close together the terms are in the document, and how important is the part of the document where they occur. However, the concept of relevancy is vague and application-specific. Different applications might require additional information for ranking, for example, document modification time. The built-in ranking functions are only examples. You can write your own ranking functions and/or combine their results with additional factors to fit your specific needs.

The two ranking functions currently available are:

```
ts_rank([ weights float4[], ] vector tsvector, query tsquery [, normalization integer ]) returns float4
```

Ranks vectors based on the frequency of their matching lexemes.

```
ts_rank_cd([ weights float4[], ] vector tsvector, query tsquery [, normalization integer ]) returns
float4
```

This function requires positional information in its input. Therefore, it will not work on "stripped" **tsvector** values. It will always return zero.

For both these functions, the optional **weights** argument offers the ability to weigh word instances more or less heavily depending on how they are labeled. The weight arrays specify how heavily to weigh each category of word, in the order:

```
{D-weight, C-weight, B-weight, A-weight}
```

If no **weights** is provided, then these defaults are used: {0.1, 0.2, 0.4, 1.0}

Typically weights are used to mark words from special areas of the document, like the title or an initial abstract, so they can be treated with more or less importance than words in the document body.

Since a longer document has a greater chance of containing a query term it is reasonable to take into account document size. For example, a hundred-word document with five instances of a search word is probably more relevant than a thousand-word document with five instances. Both ranking functions take an integer **normalization** option that specifies whether and how a document's length should impact its rank. The integer option controls several behaviors, so it is a bit mask: you can specify one or more behaviors using a vertical bar (**|**) (for example, **2|4**).

- **0** (default) ignores the document length.
- **1** divides the rank by (1 + logarithm of the document length).
- **2** divides the rank by the document length.
- **4** divides the rank by the mean harmonic distance between extents. This is implemented only by **ts_rank_cd**.
- **8** divides the rank by the number of unique words in document.
- **16** divides the rank by (1 + Logarithm of the number of unique words in document).
- **32** divides the rank by (itself + 1).

If more than one flag bit is specified, the transformations are applied in the order listed.

It is important to note that the ranking functions do not use any global information, so it is impossible to produce a fair normalization to 1% or 100% as sometimes desired. Normalization option 32 (**rank/(rank+1)**) can be applied to scale all ranks into the range zero to one, but of course this is just a cosmetic change; it will not affect the ordering of the search results.

Here is an example that selects only the ten highest-ranked matches:

```
openGauss=# SELECT id, title, ts_rank_cd(to_tsvector(body), query) AS rank
FROM tsearch.pgweb, to_tsquery('america') query
WHERE query @@ to_tsvector(body)
ORDER BY rank DESC
LIMIT 10;
 id |  title  | rank
----+---------+------
  2 | America |   .1
 11 | Brazil  |   .2
 12 | Canada  |   .1
 13 | Mexico  |   .1
(4 rows)
```

This is the same example using normalized ranking:

```
openGauss=# SELECT id, title, ts_rank_cd(to_tsvector(body), query, 32 /* rank/(rank+1) */ ) AS rank
FROM tsearch.pgweb, to_tsquery('america') query
WHERE  query @@ to_tsvector(body)
ORDER BY rank DESC
LIMIT 10;
 id |  title  |   rank
----+---------+----------
  2 | America | .0909091
 11 | Brazil  |  .166667
 12 | Canada  | .0909091
 13 | Mexico  | .0909091
(4 rows)
```

The following example sorts query by Chinese word segmentation:

```
openGauss=# CREATE TABLE tsearch.ts_ngram(id int, body text);
openGauss=# INSERT INTO tsearch.ts_ngram VALUES (1, 'Chinese');
```

```
openGauss=# INSERT INTO tsearch.ts_ngram VALUES (2, 'Chinese search');
openGauss=# INSERT INTO tsearch.ts_ngram VALUES (3 'Search Chinese');
-- Exact match
openGauss=# SELECT id, body, ts_rank_cd(to_tsvector('ngram',body), query) AS rank FROM
tsearch.ts_ngram, to_tsquery('Chinese') query WHERE query @@ to_tsvector(body);
 id | body | rank
----+------+------
  1 | Chinese |   .1
(1 row)

-- Fuzzy Match
openGauss=# SELECT id, body, ts_rank_cd(to_tsvector('ngram',body), query) AS rank FROM
tsearch.ts_ngram, to_tsquery('Chinese') query WHERE query @@ to_tsvector('ngram',body);
 id |   body   | rank
----+----------+------
  1 | Chinese     |   .1
  2 | Chinese search |   .1
  3 | Search Chinese |   .1
(3 rows)
```

Ranking can be expensive since it requires consulting the **tsvector** of each matching document, which can be I/O bound and therefore slow. Unfortunately, it is almost impossible to avoid since practical queries often result in large numbers of matches.

11.8.3.4 Highlighting Results

To present search results it is ideal to show a part of each document and how it is related to the query. Usually, search engines show fragments of the document with marked search terms. GaussDB provides a function **ts_headline** that implements this functionality.

```
ts_headline([ config regconfig, ] document text, query tsquery [, options text ]) returns text
```

ts_headline accepts a document along with a query, and returns an excerpt from the document in which terms from the query are highlighted. The configuration to be used to parse the document can be specified by **config**. If **config** is omitted, the **default_text_search_config** configuration is used.

If an options string is specified it must consist of a comma-separated list of one or more **option=value** pairs. The available options are:

- **StartSel**, **StopSel**: The strings with which to delimit query words appearing in the document, to distinguish them from other excerpted words. You must double-quote these strings if they contain spaces or commas.
- **MaxWords**, **MinWords**: These numbers determine the longest and shortest headlines to output.
- **ShortWord**: Words of this length or less will be dropped at the start and end of a headline. The default value of three eliminates common English articles.
- **HighlightAll**: Boolean flag. If **true** the whole document will be used as the headline, ignoring the preceding three parameters.
- **MaxFragments**: Maximum number of text excerpts or fragments to display. The default value of zero selects a non-fragment-oriented headline generation method. A value greater than zero selects fragment-based headline generation. This method finds text fragments with as many query words as possible and stretches those fragments around the query words. As a result query words are close to the middle of each fragment and have words on each side. Each fragment will be of at most **MaxWords** and words of length **ShortWord** or less are dropped at the start and end of each fragment.

If not all query words are found in the document, then a single fragment of the first **MinWords** in the document will be displayed.

- **FragmentDelimiter**: When more than one fragment is displayed, the fragments will be separated by this string.

Any unspecified options receive these defaults:

```
StartSel=<b>, StopSel=</b>,
MaxWords=35, MinWords=15, ShortWord=3, HighlightAll=FALSE,
MaxFragments=0, FragmentDelimiter=" ... "
```

Example:

```
openGauss=# SELECT ts_headline('english',
'The most common type of search
is to find all documents containing given query terms
and return them in order of their similarity to the
query.',
to_tsquery('english', 'query & similarity'));
                        ts_headline
------------------------------------------------------------
 containing given <b>query</b> terms
 and return them in order of their <b>similarity</b> to the
 <b>query</b>.
(1 row)

openGauss=# SELECT ts_headline('english',
'The most common type of search
is to find all documents containing given query terms
and return them in order of their similarity to the
query.',
to_tsquery('english', 'query & similarity'),
'StartSel = <, StopSel = >');
                      ts_headline
-------------------------------------------------------
 containing given <query> terms
 and return them in order of their <similarity> to the
 <query>.
(1 row)
```

ts_headline uses the original document, not a **tsvector** summary, so it can be slow and should be used with care.

11.8.4 Additional Features

11.8.4.1 Manipulating tsvector

GaussDB provides functions and operators that can be used to manipulate documents that are already in **tsvector** type.

- tsvector || tsvector

  The tsvector concatenation operator returns a new tsvector which combines the lexemes and positional information of the two tsvectors given as arguments. Positions and weight labels are retained during the concatenation. Positions appearing in the right-hand tsvector are offset by the largest position mentioned in the left-hand tsvector, so that the result is nearly equivalent to the result of performing **to_tsvector** on the concatenation of the two original document strings. (The equivalence is not exact, because any stop-words removed from the end of the left-hand argument will not affect the result, whereas they would have affected the positions of the lexemes in the right-hand argument if textual concatenation were used.)

One advantage of using concatenation in the tsvector form, rather than concatenating text before applying **to_tsvector**, is that you can use different configurations to parse different sections of the document. Also, because the **setweight** function marks all lexemes of the given tsvector the same way, it is necessary to parse the text and do **setweight** before concatenating if you want to label different parts of the document with different weights.

- setweight(vector tsvector, weight "char") returns tsvector

  setweight returns a copy of the input tsvector in which every position has been labeled with the given weight, either **A**, **B**, **C**, or **D**. (**D** is the default for new tsvectors and as such is not displayed on output.) These labels are retained when tsvectors are concatenated, allowing words from different parts of a document to be weighted differently by ranking functions.

  NOTICE

  Note that weight labels apply to positions, not lexemes. If the input tsvector has been stripped of positions then **setweight** does nothing.

- length(vector tsvector) returns integer

  Returns the number of lexemes stored in the tsvector.

- strip(vector tsvector) returns tsvector

  Returns a tsvector which lists the same lexemes as the given tsvector, but which lacks any position or weight information. While the returned tsvector is much less useful than an unstripped tsvector for relevance ranking, it will usually be much smaller.

11.8.4.2 Manipulating Queries

GaussDB provides functions and operators that can be used to manipulate queries that are already in **tsquery** type.

- tsquery && tsquery

  Returns the AND-combination of the two given queries.

- tsquery || tsquery

  Returns the OR-combination of the two given queries.

- !! tsquery

  Returns the negation (NOT) of the given query.

- numnode(query tsquery) returns integer

  Returns the number of nodes (lexemes plus operators) in a **tsquery**. This function is useful to determine if the query is meaningful (returns > 0), or contains only stop words (returns 0). Example:

```
openGauss=# SELECT numnode(plainto_tsquery('the any'));
NOTICE:  text-search query contains only stop words or doesn't contain lexemes, ignored
CONTEXT:  referenced column: numnode
 numnode
---------
       0

openGauss=# SELECT numnode('foo & bar'::tsquery);
 numnode
```

```
---------
       3
```

- querytree(query tsquery) returns text

  Returns the portion of a **tsquery** that can be used for searching an index. This function is useful for detecting non-index queries, for example those containing only stop words or only negated terms. Example:

  ```
  openGauss=# SELECT querytree(to_tsquery('!defined'));
   querytree
  -----------
   T
  (1 row)
  ```

11.8.4.3 Rewriting Queries

The **ts_rewrite** family of functions searches a given **tsquery** for occurrences of a target subquery, and replace each occurrence with a substitute subquery. In essence this operation is a **tsquery** specific version of substring replacement. A target and substitute combination can be thought of as a query rewrite rule. A collection of such rewrite rules can be a powerful search aid. For example, you can expand the search using synonyms (that is, new york, big apple, nyc, gotham) or narrow the search to direct the user to some hot topic.

- ts_rewrite (query tsquery, target tsquery, substitute tsquery) returns tsquery

  This form of **ts_rewrite** simply applies a single rewrite rule: **target** is replaced by **substitute** wherever it appears in query. Example:

  ```
  openGauss=# SELECT ts_rewrite('a & b'::tsquery, 'a'::tsquery, 'c'::tsquery);
   ts_rewrite
  ------------
   'b' & 'c'
  ```

- ts_rewrite (query tsquery, select text) returns tsquery

  This form of **ts_rewrite** accepts a starting query and a SQL select command, which is given as a text string. The **select** must yield two columns of **tsquery** type. For each row of the select result, occurrences of the first column value (the target) are replaced by the second column value (the substitute) within the current **query** value.

  NOTE

  Note that when multiple rewrite rules are applied in this way, the order of application can be important; so in practice you will want the source query to **ORDER BY** some ordering key.

  Consider a real-life astronomical example. We will expand query supernovae using table-driven rewriting rules:

  ```
  openGauss=# CREATE TABLE tsearch.aliases (id int, t tsquery, s tsquery);

  openGauss=# INSERT INTO tsearch.aliases VALUES(1, to_tsquery('supernovae'),
  to_tsquery('supernovae|sn'));

  openGauss=# SELECT ts_rewrite(to_tsquery('supernovae & crab'), 'SELECT t, s FROM tsearch.aliases');

             ts_rewrite
  ---------------------------------
   'crab' & ( 'supernova' | 'sn' )
  ```

  We can change the rewriting rules just by updating the table:

  ```
  openGauss=# UPDATE tsearch.aliases
  SET s = to_tsquery('supernovae|sn & !nebulae')
  WHERE t = to_tsquery('supernovae');
  ```

```
openGauss=# SELECT ts_rewrite(to_tsquery('supernovae & crab'), 'SELECT t, s FROM tsearch.aliases');

                ts_rewrite
---------------------------------------------
 'crab' & ( 'supernova' | 'sn' & !'nebula' )
```

Rewriting can be slow when there are many rewriting rules, since it checks every rule for a possible match. To filter out obvious non-candidate rules we can use the containment operators for the **tsquery** type. In the example below, we select only those rules which might match the original query:

```
openGauss=# SELECT ts_rewrite('a & b'::tsquery, 'SELECT t,s FROM tsearch.aliases WHERE ''a & b''::tsquery @> t');

 ts_rewrite
------------
 'b' & 'a'
(1 row)
openGauss=# DROP TABLE tsearch.aliases;
```

11.8.4.4 Gathering Document Statistics

The function **ts_stat** is useful for checking your configuration and for finding stop-word candidates.

```
ts_stat(sqlquery text, [ weights text, ]
        OUT word text, OUT ndoc integer,
        OUT nentry integer) returns setof record
```

sqlquery is a text value containing an SQL query which must return a single **tsvector** column. **ts_stat** executes the query and returns statistics about each distinct lexeme (word) contained in the **tsvector** data. The columns returned are:

- **word text**: the value of a lexeme
- **ndoc integer**: number of documents (**tsvector**) the word occurred in
- **nentry integer**: total number of occurrences of the word

If **weights** is supplied, only occurrences having one of those weights are counted. For example, to find the ten most frequent words in a document collection:

```
openGauss=# SELECT * FROM ts_stat('SELECT to_tsvector(''english'', sr_reason_sk) FROM tpcds.store_returns WHERE sr_customer_sk < 10') ORDER BY nentry DESC, ndoc DESC, word LIMIT 10;
   word | ndoc | nentry
------+------+--------
 32   |    2 |      2
 33   |    2 |      2
 1    |    1 |      1
 10   |    1 |      1
 13   |    1 |      1
 14   |    1 |      1
 15   |    1 |      1
 17   |    1 |      1
 20   |    1 |      1
 22   |    1 |      1
(10 rows)
```

The same, but counting only word occurrences with weight **A** or **B**:

```
openGauss=# SELECT * FROM ts_stat('SELECT to_tsvector(''english'', sr_reason_sk) FROM tpcds.store_returns WHERE sr_customer_sk < 10', 'a') ORDER BY nentry DESC, ndoc DESC, word LIMIT 10;
 word | ndoc | nentry
------+------+--------
(0 rows)
```

11.8.5 Parser

Text search parsers are responsible for splitting raw document text into tokens and identifying each token's type, where the type set is defined by the parser itself. Note that a parser does not modify the text; it simply identifies plausible word boundaries. Because of this limit, there is less need for application-specific custom parsers than there is for custom dictionaries.

Currently, GaussDB provides the following built-in parsers: pg_catalog.default for English configuration, and pg_catalog.ngram and pg_catalog.pound for full text search in texts containing Chinese, or both Chinese and English.

The built-in parser is named **pg_catalog.default**. It recognizes 23 token types, shown in **Table 11-92**.

Table 11-92 Token types of the default parser

Alias	Description	Example
asciiword	Word, all ASCII letters	elephant
word	Word, all letters	mañana
numword	Word, letters and digits	beta1
asciihword	Hyphenated word, all ASCII	up-to-date
hword	Hyphenated word, all letters	lógico-matemática
numhword	Hyphenated word, letters and digits	openGauss-beta1
hword_asciipart	Hyphenated word part, all ASCII	openGauss in the context openGauss-beta1
hword_part	Hyphenated word part, all letters	lógico or matemática in the context lógico-matemática
hword_numpart	Hyphenated word part, letters and digits	beta1 in the context openGauss-beta1
email	Email address	foo@example.com
protocol	Protocol head	http://
url	URL	example.com/stuff/index.html
host	Host	example.com
url_path	URL path	/stuff/index.html, in the context of a URL
file	File or path name	/usr/local/foo.txt, if not within a URL
sfloat	Scientific notation	-1.23E+56

Alias	Description	Example
float	Decimal notation	-1.234
int	Signed integer	-1234
uint	Unsigned integer	1234
version	Version number	8.3.0
tag	XML tag	<a href="dictionaries.html">
entity	XML entity	&amp;
blank	Space symbols	(any whitespace or punctuation not otherwise recognized)

Note: The parser's notion of a "letter" is determined by the database's locale setting, specifically **lc_ctype**. Words containing only the basic ASCII letters are reported as a separate token type, since it is sometimes useful to distinguish them. In most European languages, token types word and asciiword should be treated alike.

email does not support all valid email characters as defined by RFC 5322. Specifically, the only non-alphanumeric characters supported for email usernames are period, dash, and underscore.

It is possible for the parser to identify overlapping tokens in the same piece of text. For example, a hyphenated word will be reported both as the entire word and as each component.

```
openGauss=# SELECT alias, description, token FROM ts_debug('english','foo-bar-beta1');
      alias      |               description                |     token
-----------------+------------------------------------------+---------------
 numhword        | Hyphenated word, letters and digits      | foo-bar-beta1
 hword_asciipart | Hyphenated word part, all ASCII          | foo
 blank           | Space symbols                            | -
 hword_asciipart | Hyphenated word part, all ASCII          | bar
 blank           | Space symbols                            | -
 hword_numpart   | Hyphenated word part, letters and digits | beta1
```

This behavior is desirable since it allows searches to work for both the whole compound word and for components. Here is another instructive example:

```
openGauss=# SELECT alias, description, token FROM ts_debug('english','http://example.com/stuff/
index.html');
  alias   |  description  |            token
----------+---------------+------------------------------
 protocol | Protocol head | http://
 url      | URL           | example.com/stuff/index.html
 host     | Host          | example.com
 url_path | URL path      | /stuff/index.html
```

N-gram is a mechanical word segmentation method, and applies to no semantic Chinese segmentation scenarios. The N-gram segmentation method ensures the completeness of the segmentation. However, to cover all the possibilities, it adds unnecessary words to the index, resulting in a large number of index items. N-gram supports Chinese coding, including GBK and UTF-8, and has six built-in token types, as shown in **Table 11-93**.

Table 11-93 Token types

Alias	Description
zh_words	chinese words
en_word	english word
numeric	numeric data
alnum	alnum string
grapsymbol	graphic symbol
multisymbol	multiple symbol

Pound segments words in a fixed format. It is used to segment to-be-parsed nonsense Chinese and English words that are separated by fixed separators. It supports Chinese encoding (including GBK and UTF8) and English encoding (including ASCII). Six built-in token types are available, as listed in **Table 11-94**. Five types of delimiters are supported, as shown in **Table 11-95**, and the default delimiter is #. The maximum length of a token is 256 characters.

Table 11-94 Token types

Alias	Description
zh_words	chinese words
en_word	english word
numeric	numeric data
alnum	alnum string
grapsymbol	graphic symbol
multisymbol	multiple symbol

Table 11-95 Separator types

Separator	Description
@	Special character
#	Special character
$	Special character
%	Special character
/	Special character

11.8.6 Dictionaries

11.8.6.1 Overview

A dictionary is used to define stop words, that is, words to be ignored in full-text retrieval.

A dictionary can also be used to normalize words so that different derived forms of the same word will match. A normalized word is called a lexeme.

In addition to improving retrieval quality, normalization and removal of stop words can reduce the size of the **tsvector** representation of a document, thereby improving performance. Normalization and removal of stop words do not always have linguistic meaning. Users can define normalization and removal rules in dictionary definition files based on application environments.

A dictionary is a program that receives a token as input and returns:

- An array of lexemes if the input token is known to the dictionary (note that one token can produce more than one lexeme).
- A single lexeme A new token is passed to the subsequent dictionary (the current dictionary may be referred to as a filter dictionary) in place of the input token.
- An empty array if the input token is known to the dictionary but is a stop word.
- **NULL** if the dictionary does not recognize the token.

GaussDB provides predefined dictionaries for many languages and also provides five predefined dictionary templates, **Simple**, **Synonym**, **Thesaurus**, **Ispell**, and **Snowball**. These templates can be used to create new dictionaries with custom parameters.

When using full-text retrieval, you are advised to:

- In the text search configuration, configure a parser together with a set of dictionaries to process the parser's output tokens. For each token type that the parser can return, a separate list of dictionaries is specified by the configuration. When a token of that type is found by the parser, each dictionary in the list is consulted in turn, until a dictionary recognizes it as a known word. If it is identified as a stop word, or no dictionary recognizes the token, it will be discarded and not indexed or searched for. Generally, the first dictionary that returns a non-**NULL** output determines the result, and any remaining dictionaries are not consulted. However, a filtering dictionary can replace the input token with a modified one, which is then passed to subsequent dictionaries.
- The general rule for configuring a list of dictionaries is to place first the most narrow, most specific dictionary, then the more general dictionaries, finishing with a very general dictionary, like a **Snowball** stemmer dictionary or a **Simple** dictionary, which recognizes everything. In the following example, for an astronomy-specific search (**astro_en** configuration), you can configure the token type **asciiword** (ASCII word) with a **Synonym** dictionary of astronomical terms, a general English **Ispell** dictionary, and a **Snowball** English stemmer dictionary:

```
openGauss=# ALTER TEXT SEARCH CONFIGURATION astro_en
  ADD MAPPING FOR asciiword WITH astro_syn, english_ispell, english_stem;
```

A filtering dictionary can be placed anywhere in the list, except at the end where it would be useless. Filtering dictionaries are useful to partially normalize words to simplify the task of later dictionaries.

11.8.6.2 Stop Words

Stop words are words that are very common, appear in almost every document, and have no discrimination value. Therefore, they can be ignored in the context of full text searching. Each type of dictionaries treats stop words in different ways. For example, **Ispell** dictionaries first normalize words and then check the list of stop words, while **Snowball** dictionaries first check the list of stop words.

For example, every English text contains words like **a** and **the**, so it is useless to store them in an index. However, stop words affect the positions in **tsvector**, which in turn affect ranking.

```
openGauss=# SELECT to_tsvector('english','in the list of stop words');
        to_tsvector
----------------------------
 'list':3 'stop':5 'word':6
```

The missing positions 1, 2, and 4 are because of stop words. Ranks calculated for documents with and without stop words are quite different:

```
openGauss=# SELECT ts_rank_cd (to_tsvector('english','in the list of stop words'), to_tsquery('list & stop'));
 ts_rank_cd
------------
        .05

openGauss=# SELECT ts_rank_cd (to_tsvector('english','list stop words'), to_tsquery('list & stop'));
 ts_rank_cd
------------
         .1
```

11.8.6.3 Simple Dictionary

A **Simple** dictionary operates by converting the input token to lower case and checking it against a list of stop words. If the token is found in the list, an empty array will be returned, causing the token to be discarded. If it is not found, the lower-cased form of the word is returned as the normalized lexeme. In addition, you can set **Accept** to **false** for **Simple** dictionaries (default: **true**) to report non-stop-words as unrecognized, allowing them to be passed on to the next dictionary in the list.

Precautions

- Most types of dictionaries rely on dictionary configuration files. The name of a configuration file can only be lowercase letters, digits, and underscores (_).
- A dictionary cannot be created in **pg_temp** mode.
- Dictionary configuration files must be stored in UTF-8 encoding. They will be translated to the actual database encoding, if that is different, when they are read into the server.
- Generally, a session will read a dictionary configuration file only once, when it is first used within the session. To modify a configuration file, run the **ALTER TEXT SEARCH DICTIONARY** statement to update and reload the file.

Procedure

Step 1 Create a **Simple** dictionary.

```
openGauss=# CREATE TEXT SEARCH DICTIONARY public.simple_dict (
    TEMPLATE = pg_catalog.simple,
    STOPWORDS = english
);
```

english.stop is the full name of a file of stop words. For details about the syntax and parameters for creating a **Simple** dictionary, see **CREATE TEXT SEARCH DICTIONARY**.

Step 2 Use the **Simple** dictionary.

```
openGauss=# SELECT ts_lexize('public.simple_dict','YeS');
 ts_lexize
-----------
 {yes}
(1 row)

openGauss=# SELECT ts_lexize('public.simple_dict','The');
 ts_lexize
-----------
 {}
(1 row)
```

Step 3 Set **Accept=false** so that the **Simple** dictionary returns **NULL** instead of a lower-cased non-stop word.

```
openGauss=# ALTER TEXT SEARCH DICTIONARY public.simple_dict ( Accept = false );
ALTER TEXT SEARCH DICTIONARY
openGauss=# SELECT ts_lexize('public.simple_dict','YeS');
 ts_lexize
-----------

(1 row)

openGauss=# SELECT ts_lexize('public.simple_dict','The');
 ts_lexize
-----------
 {}
(1 row)
```

----End

11.8.6.4 Synonym Dictionary

A **Synonym** dictionary is used to define, identify, and convert synonyms of a token. Phrases are not supported. Synonyms of phrases can be defined in a **Thesaurus** dictionary. For details, see **Thesaurus Dictionary**.

Examples

- A **Synonym** dictionary can be used to overcome linguistic problems. For example, to prevent an English stemmer dictionary from reducing the word 'Paris' to 'pari', define a **Paris paris** line in the **Synonym** dictionary and put it before the **english_stem** dictionary.

```
openGauss=# SELECT * FROM ts_debug('english', 'Paris');
   alias   |   description   | token |  dictionaries  |  dictionary  | lexemes
-----------+-----------------+-------+----------------+--------------+---------
 asciiword | Word, all ASCII | Paris | {english_stem} | english_stem | {pari}
(1 row)

openGauss=# CREATE TEXT SEARCH DICTIONARY my_synonym (
    TEMPLATE = synonym,
    SYNONYMS = my_synonyms,
    FILEPATH = 'file:///home/dicts/'
);

openGauss=# ALTER TEXT SEARCH CONFIGURATION english
    ALTER MAPPING FOR asciiword
    WITH my_synonym, english_stem;

openGauss=# SELECT * FROM ts_debug('english', 'Paris');
   alias   |   description   | token |       dictionaries        | dictionary | lexemes
-----------+-----------------+-------+---------------------------+------------+---------
 asciiword | Word, all ASCII | Paris | {my_synonym,english_stem} | my_synonym | {paris}
(1 row)

openGauss=# SELECT * FROM ts_debug('english', 'paris');
   alias   |   description   | token |       dictionaries        | dictionary | lexemes
-----------+-----------------+-------+---------------------------+------------+---------
 asciiword | Word, all ASCII | Paris | {my_synonym,english_stem} | my_synonym | {paris}
(1 row)

openGauss=# ALTER TEXT SEARCH DICTIONARY my_synonym ( CASESENSITIVE=true);

openGauss=# SELECT * FROM ts_debug('english', 'Paris');
   alias   |   description   | token |       dictionaries        | dictionary | lexemes
-----------+-----------------+-------+---------------------------+------------+---------
 asciiword | Word, all ASCII | Paris | {my_synonym,english_stem} | my_synonym | {paris}
(1 row)

openGauss=# SELECT * FROM ts_debug('english', 'paris');
   alias   |   description   | token |       dictionaries        | dictionary | lexemes
-----------+-----------------+-------+---------------------------+------------+---------
 asciiword | Word, all ASCII | Paris | {my_synonym,english_stem} | my_synonym | {pari}
(1 row)
```

The full name of the **Synonym** dictionary file is **my_synonyms.syn**, and the dictionary is stored in the **/home/dicts/** directory of the current database primary node. For details about the syntax and parameters for creating an **Ispell** dictionary, see **ALTER TEXT SEARCH DICTIONARY**.

- An asterisk (*) can be placed at the end of a synonym in the configuration file. This indicates that the synonym is a prefix. The asterisk is ignored when the entry is used in **to_tsvector()**, but when it is used in **to_tsquery()**, the result will be a query item with the prefix match marker (see **Manipulating Queries**).

  Assume that the content in the dictionary file **synonym_sample.syn** is as follows:

```
postgres        pgsql
postgresql      pgsql
postgre pgsql
gogle   googl
indices index*
```

  Create and use a dictionary.

```
openGauss=# CREATE TEXT SEARCH DICTIONARY syn (
    TEMPLATE = synonym,
    SYNONYMS = synonym_sample
);

openGauss=# SELECT ts_lexize('syn','indices');
 ts_lexize
-----------
 {index}
(1 row)

openGauss=# CREATE TEXT SEARCH CONFIGURATION tst (copy=simple);

openGauss=# ALTER TEXT SEARCH CONFIGURATION tst ALTER MAPPING FOR asciiword WITH syn;
```

```
openGauss=# SELECT to_tsvector('tst','indices');
 to_tsvector
-------------
 'index':1
(1 row)

openGauss=# SELECT to_tsquery('tst','indices');
 to_tsquery
------------
 'index':*
(1 row)

openGauss=# SELECT 'indexes are very useful'::tsvector;
            tsvector
---------------------------------
 'are' 'indexes' 'useful' 'very'
(1 row)

openGauss=# SELECT 'indexes are very useful'::tsvector @@ to_tsquery('tst','indices');
 ?column?
----------
 t
(1 row)
```

11.8.6.5 Thesaurus Dictionary

A **Thesaurus** dictionary (sometimes abbreviated as TZ) is a collection of relationships between words and phrases, such as broader terms (BT), narrower terms (NT), preferred terms, non-preferred terms, and related terms. Based on definitions in the dictionary file, a TZ replaces all non-preferred terms by one preferred term and, optionally, preserves the original terms for indexing as well. A TZ is an extension of a **Synonym** dictionary with added phrase support.

Precautions

- A TZ has the capability to recognize phrases and therefore it must remember its state and interact with the parser to determine whether to handle the next token or stop accumulation. A TZ must be configured carefully. For example, if a TZ is configured to handle only **asciiword** tokens, a TZ definition like **one 7** will not work because the token type **uint** is not assigned to the TZ.
- TZs are used during indexing, so any change in the TZ's parameters requires reindexing. For most other dictionary types, small changes such as adding or removing stop words does not force reindexing.

Procedure

Step 1 Create a TZ named **thesaurus_astro**.

thesaurus_astro is a simple astronomical TZ that defines two astronomical word combinations (word+synonym).

```
supernovae stars : sn
crab nebulae : crab
```

Run the following statement to create the TZ:

```
openGauss=# CREATE TEXT SEARCH DICTIONARY thesaurus_astro (
    TEMPLATE = thesaurus,
    DictFile = thesaurus_astro,
    Dictionary = pg_catalog.english_stem,
    FILEPATH = 'file:///home/dicts/'
);
```

The full name of the TZ file is **thesaurus_astro.ths**, and the TZ is stored in the **/home/dicts/** directory of the current database primary node. **pg_catalog.english_stem** is the subdictionary (a **Snowball** English stemmer) used for input normalization. The subdictionary has its own configuration (for example, stop words), which is not shown here. For details about the syntax and parameters for creating an **Ispell** dictionary, see **CREATE TEXT SEARCH DICTIONARY**.

Step 2 Bind the TZ to the desired token types in the text search configuration.

```
openGauss=# ALTER TEXT SEARCH CONFIGURATION russian
   ALTER MAPPING FOR asciiword, asciihword, hword_asciipart
   WITH thesaurus_astro, english_stem;
```

Step 3 Use the TZ.

- Test the TZ.

  The **ts_lexize** function is not very useful for testing the TZ because the function processes its input as a single token. Instead, you can use the **plainto_tsquery**, **to_tsvector**, or **to_tsquery** function which will break their input strings into multiple tokens.

```
openGauss=# SELECT plainto_tsquery('russian','supernova star');
 plainto_tsquery
-----------------
 'sn'
(1 row)

openGauss=# SELECT to_tsvector('russian','supernova star');
 to_tsvector
-------------
 'sn':1
(1 row)

openGauss=# SELECT to_tsquery('russian','''supernova star''');
 to_tsquery
------------
 'sn'
(1 row)
```

  supernova star matches **supernovae stars** in **thesaurus_astro** because the Snowball **english_stem** stemmer is specified in the **thesaurus_astro** definition. The stemmer removed **e** and **s**.

- To index the original phrase, include it in the right-hand part of the definition.

```
supernovae stars : sn supernovae stars

openGauss=# ALTER TEXT SEARCH DICTIONARY thesaurus_astro (
    DictFile = thesaurus_astro,
    FILEPATH = 'file:///home/dicts/');

openGauss=# SELECT plainto_tsquery('russian','supernova star');
      plainto_tsquery
-----------------------------
 'sn' & 'supernova' & 'star'
(1 row)
```

----End

11.8.6.6 Ispell Dictionary

An **Ispell** dictionary is a morphological dictionary, which can normalize different linguistic forms of a word into the same lexeme. For example, an English **Ispell** dictionary can match all declensions and conjugations of the search term **bank**, such as, **banking**, **banked**, **banks**, **banks'**, and **bank's**.

GaussDB does not provide any predefined **Ispell** dictionaries or dictionary files. The .dict files and .affix files support multiple open-source dictionary formats, including **Ispell**, **MySpell**, and **Hunspell**.

Procedure

Step 1 Obtain the dictionary definition file (.dict) and affix file (.affix).

You can use an open-source dictionary. The name extensions of the open-source dictionary may be .aff and .dic. In this case, you need to change them to .affix and .dict. In addition, for some dictionary files (for example, Norwegian dictionary files), you need to run the following commands to convert the character encoding to UTF-8:

```
iconv -f ISO_8859-1 -t UTF-8 -o nn_no.affix nn_NO.aff
iconv -f ISO_8859-1 -t UTF-8 -o nn_no.dict nn_NO.dic
```

Step 2 Create an **Ispell** dictionary.

```
openGauss=# CREATE TEXT SEARCH DICTIONARY norwegian_ispell (
    TEMPLATE = ispell,
    DictFile = nn_no,
    AffFile = nn_no,
    FilePath = 'file:///home/dicts'
);
```

The full names of the **Ispell** dictionary files are **nn_no.dict** and **nn_no.affix**, and the dictionary is stored in the **/home/dicts/** directory of the current database primary node. For details about the syntax and parameters for creating a dictionary, see **CREATE TEXT SEARCH DICTIONARY**.

Step 3 Use the **Ispell** dictionary to split compound words.

```
openGauss=# SELECT ts_lexize('norwegian_ispell', 'sjokoladefabrikk');
      ts_lexize
---------------------
 {sjokolade,fabrikk}
(1 row)
```

MySpell does not support compound words. **Hunspell** supports compound words. GaussDB supports only the basic compound word operations of **Hunspell**. Generally, **Ispell** dictionaries recognize a limited set of words, so they should be followed by another broader dictionary, for example, a **Snowball** dictionary, which recognizes everything.

----End

11.8.6.7 Snowball Dictionary

A **Snowball** dictionary is based on a project by Martin Porter and is used for stem analysis, providing stemming algorithms for many languages. GaussDB provides predefined **Snowball** dictionaries of many languages. You can query the **PG_TS_DICT** system catalog to view the predefined **Snowball** dictionaries and supported stemming algorithms.

A **Snowball** dictionary recognizes everything, no matter whether it is able to simplify the word. Therefore, it should be placed at the end of the dictionary list. It is useless to place it before any other dictionary because a token will never pass it through to the next dictionary.

For details about the syntax of **Snowball** dictionaries, see **CREATE TEXT SEARCH DICTIONARY**.

11.8.7 Configuration Examples

Text search configuration specifies the following components required for converting a document into a **tsvector**:

- A parser, decomposes a text into tokens.
- Dictionary list, converts each token into a lexeme.

Each time when the **to_tsvector** or **to_tsquery** function is invoked, a text search configuration is required to specify a processing procedure. The GUC parameter **default_text_search_config** specifies the default text search configuration, which will be used if the text search function does not explicitly specify a text search configuration.

GaussDB provides some predefined text search configurations. You can also create user-defined text search configurations. In addition, to facilitate the management of text search objects, multiple **gsql** meta-commands are provided to display information about text search objects. For details, see "Client Tool > Meta-Command Reference" in *Tool Reference*.

Procedure

Step 1 Create a text search configuration **ts_conf** by copying the predefined text search configuration **english**.

```
openGauss=# CREATE TEXT SEARCH CONFIGURATION ts_conf ( COPY = pg_catalog.english );
CREATE TEXT SEARCH CONFIGURATION
```

Step 2 Create a **Synonym** dictionary.

Assume that the definition file **pg_dict.syn** of the **Synonym** dictionary contains the following contents:

```
postgres    pg
pgsql       pg
postgresql  pg
```

Run the following statement to create the **Synonym** dictionary:

```
openGauss=# CREATE TEXT SEARCH DICTIONARY pg_dict (
     TEMPLATE = synonym,
     SYNONYMS = pg_dict,
     FILEPATH = 'file:///home/dicts'
 );
```

Step 3 Create an **Ispell** dictionary **english_ispell** (the dictionary definition file is from the open source dictionary).

```
openGauss=# CREATE TEXT SEARCH DICTIONARY english_ispell (
    TEMPLATE = ispell,
    DictFile = english,
    AffFile = english,
    StopWords = english,
    FILEPATH = 'file:///home/dicts'
);
```

Step 4 Modify the text search configuration **ts_conf** and change the dictionary list for tokens of certain types. For details about token types, see **Parser**.

```
openGauss=# ALTER TEXT SEARCH CONFIGURATION ts_conf
    ALTER MAPPING FOR asciiword, asciihword, hword_asciipart,
                    word, hword, hword_part
    WITH pg_dict, english_ispell, english_stem;
```

Step 5 In the text search configuration, set non-index or set the search for tokens of certain types.

```
openGauss=# ALTER TEXT SEARCH CONFIGURATION ts_conf
    DROP MAPPING FOR email, url, url_path, sfloat, float;
```

Step 6 Use the text retrieval commissioning function **ts_debug()** to test the text search configuration **ts_conf**.

```
openGauss=# SELECT * FROM ts_debug('ts_conf', '
PostgreSQL, the highly scalable, SQL compliant, open source object-relational
database management system, is now undergoing beta testing of the next
version of our software.
');
```

Step 7 You can set the default text search configuration of the current session to **ts_conf**. This setting is valid only for the current session.

```
openGauss=# \dF+ ts_conf
      Text search configuration "public.ts_conf"
Parser: "pg_catalog.default"
      Token      |            Dictionaries
-----------------+-------------------------------------
 asciihword      | pg_dict,english_ispell,english_stem
 asciiword       | pg_dict,english_ispell,english_stem
 file            | simple
 host            | simple
 hword           | pg_dict,english_ispell,english_stem
 hword_asciipart | pg_dict,english_ispell,english_stem
 hword_numpart   | simple
 hword_part      | pg_dict,english_ispell,english_stem
 int             | simple
 numhword        | simple
 numword         | simple
 uint            | simple
 version         | simple
 word            | pg_dict,english_ispell,english_stem

openGauss=# SET default_text_search_config = 'public.ts_conf';
SET
openGauss=# SHOW default_text_search_config;
 default_text_search_config
----------------------------
 public.ts_conf
(1 row)
```

----End

11.8.8 Testing and Debugging Text Search

11.8.8.1 Testing a Configuration

The **ts_debug** function allows easy testing of a text search configuration.

```
ts_debug([ config regconfig, ] document text,
        OUT alias text,
        OUT description text,
        OUT token text,
        OUT dictionaries regdictionary[],
        OUT dictionary regdictionary,
        OUT lexemes text[])
        returns setof record
```

ts_debug displays information about every token of document as produced by the parser and processed by the configured dictionaries. It uses the configuration specified by **config**, or **default_text_search_config** if that argument is omitted.

ts_debug returns one row for each token identified in the text by the parser. The columns returned are:

- **alias text**: short name of the token type
- **description text**: description of the token type
- **token text**: text of the token
- **dictionaries regdictionary[]**: dictionaries selected by the configuration for this token type
- **dictionary regdictionary**: the dictionary that recognized the token, or NULL if none did
- **lexemes text[]**: the lexeme(s) produced by the dictionary that recognized the token, or NULL if none did; an empty array ({}) means the token was recognized as a stop word

Here is a simple example:

```
openGauss=# SELECT * FROM ts_debug('english','a fat  cat sat on a mat - it ate a fat rats');
   alias   |   description   | token |  dictionaries  |  dictionary  | lexemes
-----------+-----------------+-------+----------------+--------------+---------
 asciiword | Word, all ASCII | a     | {english_stem} | english_stem | {}
 blank     | Space symbols   |       | {}             |              |
 asciiword | Word, all ASCII | fat   | {english_stem} | english_stem | {fat}
 blank     | Space symbols   |       | {}             |              |
 asciiword | Word, all ASCII | cat   | {english_stem} | english_stem | {cat}
 blank     | Space symbols   |       | {}             |              |
 asciiword | Word, all ASCII | sat   | {english_stem} | english_stem | {sat}
 blank     | Space symbols   |       | {}             |              |
 asciiword | Word, all ASCII | on    | {english_stem} | english_stem | {}
 blank     | Space symbols   |       | {}             |              |
 asciiword | Word, all ASCII | a     | {english_stem} | english_stem | {}
 blank     | Space symbols   |       | {}             |              |
 asciiword | Word, all ASCII | mat   | {english_stem} | english_stem | {mat}
 blank     | Space symbols   |       | {}             |              |
 blank     | Space symbols   | -     | {}             |              |
 asciiword | Word, all ASCII | it    | {english_stem} | english_stem | {}
 blank     | Space symbols   |       | {}             |              |
 asciiword | Word, all ASCII | ate   | {english_stem} | english_stem | {ate}
 blank     | Space symbols   |       | {}             |              |
 asciiword | Word, all ASCII | a     | {english_stem} | english_stem | {}
 blank     | Space symbols   |       | {}             |              |
 asciiword | Word, all ASCII | fat   | {english_stem} | english_stem | {fat}
 blank     | Space symbols   |       | {}             |              |
 asciiword | Word, all ASCII | rats  | {english_stem} | english_stem | {rat}
(24 rows)
```

11.8.8.2 Testing a Parser

The **ts_parse** function allows direct testing of a text search parser.

```
ts_parse(parser_name text, document text,
        OUT tokid integer, OUT token text) returns setof record
```

ts_parse parses the given **document** and returns a series of records, one for each token produced by parsing. Each record includes a **tokid** showing the assigned token type and a **token** which is the text of the token. Example:

```
openGauss=# SELECT * FROM ts_parse('default', '123 - a number');
 tokid | token
```

```
-------+--------
    22 | 123
    12 |
    12 | -
     1 | a
    12 |
     1 | number
(6 rows)
```

The **ts_token_type** function returns the token type and description of the specified parser.

```
ts_token_type(parser_name text, OUT tokid integer,
              OUT alias text, OUT description text) returns setof record
```

ts_token_type returns a table which describes each type of token the specified parser can recognize. For each token type, the table gives the integer **tokid** that the parser uses to label a token of that type, the **alias** that names the token type in configuration commands, and a short description. Example:

```
openGauss=# SELECT * FROM ts_token_type('default');
 tokid |      alias      |               description
-------+-----------------+------------------------------------------
     1 | asciiword       | Word, all ASCII
     2 | word            | Word, all letters
     3 | numword         | Word, letters and digits
     4 | email           | Email address
     5 | url             | URL
     6 | host            | Host
     7 | sfloat          | Scientific notation
     8 | version         | Version number
     9 | hword_numpart   | Hyphenated word part, letters and digits
    10 | hword_part      | Hyphenated word part, all letters
    11 | hword_asciipart | Hyphenated word part, all ASCII
    12 | blank           | Space symbols
    13 | tag             | XML tag
    14 | protocol        | Protocol head
    15 | numhword        | Hyphenated word, letters and digits
    16 | asciihword      | Hyphenated word, all ASCII
    17 | hword           | Hyphenated word, all letters
    18 | url_path        | URL path
    19 | file            | File or path name
    20 | float           | Decimal notation
    21 | int             | Signed integer
    22 | uint            | Unsigned integer
    23 | entity          | XML entity
(23 rows)
```

11.8.8.3 Testing a Dictionary

The **ts_lexize** function facilitates dictionary testing.

ts_lexize(dict regdictionary, token text) returns text[] ts_lexize returns an array of lexemes if the input **token** is known to the dictionary, or an empty array if the token is known to the dictionary but it is a stop word, or **NULL** if it is an unknown word.

Example:

```
openGauss=# SELECT ts_lexize('english_stem', 'stars');
 ts_lexize
-----------
 {star}

openGauss=# SELECT ts_lexize('english_stem', 'a');
 ts_lexize
```

```
-----------
{}
```

NOTICE

The **ts_lexize** function expects a single **token**, not text.

11.8.9 Limitations

The current limitations of GaussDB's text search features are:

- The length of each lexeme must be less than 2 KB.
- The length of a **tsvector** (lexemes + positions) must be less than 1 megabyte.
- Position values in **tsvector** must be greater than 0 and no more than 16383.
- No more than 256 positions per lexeme. Excessive positions, if any, will be discarded.
- The number of nodes (lexemes + operators) in a tsquery must be less than 32768.

11.9 System Operation

GaussDB text runs SQL statements to perform different system operations, such as setting variables, displaying the execution plan, and collecting garbage data.

Setting Variables

For details about how to set various parameters for a session or transaction, see **SET**.

Displaying the Execution Plan

For details about how to display the execution plan that GaussDB makes for SQL statements, see **EXPLAIN**.

Specifying a Checkpoint in Transaction Logs

By default, WALs periodically specify checkpoints in a transaction log. **CHECKPOINT** forces an immediate checkpoint when the related command is issued, without waiting for a regular checkpoint scheduled by the system. For details, see **CHECKPOINT**.

Collecting Unnecessary Data

For details about how to collect garbage data and analyze a database as required, see **VACUUM**.

Collecting Statistics

For details about how to collect statistics on tables in databases, see **ANALYZE | ANALYSE**.

Setting the Constraint Check Mode for the Current Transaction

For details about how to set the constraint check mode for the current transaction, see **SET CONSTRAINTS**.

Shutting Down The Current Database Node

For details about shutting down the current database node, see **SHUTDOWN**.

11.10 Controlling Transactions

A transaction is a user-defined sequence of database operations, which form an integral unit of work.

Starting a Transaction

GaussDB starts a transaction using **START TRANSACTION** and **BEGIN**. For details, see **START TRANSACTION** and **BEGIN**.

Setting a Transaction

GaussDB sets a transaction using **SET TRANSACTION** or **SET LOCAL TRANSACTION**. For details, see **SET TRANSACTION**.

Committing a Transaction

GaussDB commits all operations of a transaction using **COMMIT** or **END**. For details, see **COMMIT | END**.

Rolling Back a Transaction

If a fault occurs during a transaction and the transaction cannot proceed, the system performs rollback to cancel all the completed database operations related to the transaction. See **ROLLBACK**.

NOTE

If an execution request (not in a transaction block) received in the database contains multiple statements, the request is packed into a transaction. If one of the statements fails, the entire request will be rolled back.

11.11 DDL Syntax Overview

Data definition language (DDL) is used to define or modify an object in a database, such as a table, an index, or a view.

NOTE

GaussDB does not support DDL when the primary node of the database is incomplete. For example, if the primary node of the database is faulty, creating a database or a table will fail.

Defining a CMK

CMKs are used to encrypt CEKs for the encrypted database feature. CMK definition includes creating and deleting a CMK. For details about related SQL statements, see **Table 11-96**.

Table 11-96 SQL statements for defining a CMK

Function	SQL Statement
Creating a CMK	**CREATE CLIENT MASTER KEY**
Deleting a CMK	**DROP CLIENT MASTER KEY**

Defining a CEK

CEKs are used to encrypt data for the encrypted database feature. CEK definition includes creating and deleting a CEK. For details about related SQL statements, see **Table 11-96**.

Table 11-97 SQL statements for defining a CEK

Function	SQL Statement
Creating a CEK	**CREATE COLUMN ENCRYPTION KEY**
Deleting a CEK	**DROP COLUMN ENCRYPTION KEY**

Defining a Database

A database is the warehouse for organizing, storing, and managing data. Defining a database includes creating a database, altering the database attributes, and deleting the database. For details about related SQL statements, see **Table 11-98**.

Table 11-98 SQL statements for defining a database

Function	SQL Statement
Creating a database	**CREATE DATABASE**
Altering a database attribute	**ALTER DATABASE**
Deleting a database	**DROP DATABASE**

Defining a schema

A schema is the set of a group of database objects and is used to control the access to the database objects. For details about related SQL statements, see **Table 11-99**.

Table 11-99 SQL statements for defining a schema

Function	SQL Statement
Creating a schema	**CREATE SCHEMA**
Altering a schema attribute	**ALTER SCHEMA**
Deleting a schema	**DROP SCHEMA**

Defining a Tablespace

A tablespace is used to manage data objects and corresponds to a catalog on a disk. For details about related SQL statements, see **Table 11-100**.

Table 11-100 SQL statements for defining a tablespace

Function	SQL Statement
Creating a tablespace	**CREATE TABLESPACE**
Altering a tablespace attribute	**ALTER TABLESPACE**
Deleting a tablespace	**DROP TABLESPACE**

Defining a Table

A table is a special data structure in a database and is used to store data objects and relationship between data objects. For details about related SQL statements, see **Table 11-101**.

Table 11-101 SQL statements for defining a table

Function	SQL Statement
Creating a table	**CREATE TABLE**
Altering a table attribute	**ALTER TABLE**
Deleting a table	**DROP TABLE**

Defining a Partitioned Table

A partitioned table is a logical table used to improve query performance and does not store data (data is stored in common tables). For details about related SQL statements, see **Table 11-102**.

Table 11-102 SQL statements for defining a partitioned table

Function	SQL Statement
Creating a partitioned table	**CREATE TABLE PARTITION**
Creating a partition	**ALTER TABLE PARTITION**
Altering a partitioned table attribute	**ALTER TABLE PARTITION**
Deleting a partition	**ALTER TABLE PARTITION**
Deleting a partitioned table	**DROP TABLE**

Defining an Index

An index indicates the sequence of values in one or more columns in a database table. It is a data structure that improves the speed of data access to specific information in a database table. For details about related SQL statements, see **Table 11-103**.

Table 11-103 SQL statements for defining an index

Function	SQL Statement
Creating an index	**CREATE INDEX**
Altering an index attribute	**ALTER INDEX**
Deleting an index	**DROP INDEX**
Rebuilding an index	**REINDEX**

Defining a Stored Procedure

A stored procedure is a set of SQL statements for achieving specific functions and is stored in the database after compiling. Users can specify a name and provide parameters (if necessary) to execute the stored procedure. For details about related SQL statements, see **Table 11-104**.

Table 11-104 SQL statements for defining a stored procedure

Function	SQL Statement
Creating a stored procedure	**CREATE PROCEDURE**
Deleting a stored procedure	**DROP PROCEDURE**

Defining a Function

In the primary node of the database, a function is similar to a stored procedure, which is a set of SQL statements. The function and stored procedure are used the same. For details about related SQL statements, see **Table 11-105**.

Table 11-105 SQL statements for defining a function

Function	SQL Statement
Creating a function	**CREATE FUNCTION**
Altering a function attribute	**ALTER FUNCTION**
Deleting a function	**DROP FUNCTION**

Defining a Package

A package consists of the package specification and package body. It is used to manage stored procedures and functions by class, which is similar to classes in languages such as Java and C++.

Table 11-106 SQL statements for defining a package

Function	SQL Statement
Creating a package	**CREATE PACKAGE**
Deleting a package	**DROP PACKAGE**
Altering a package attribute	**ALTER PACKAGE**

Defining a View

A view is a virtual table exported from one or more basic tables. It is used to control data accesses of users. **Table 11-107** lists the related SQL statements.

Table 11-107 SQL statements for defining a view

Function	SQL Statement
Creating a view	**CREATE VIEW**
Deleting a view	**DROP VIEW**

Defining a Cursor

To process SQL statements, the stored procedure process assigns a memory segment to store context association. Cursors are handles or pointers to context regions. With a cursor, the stored procedure can control alterations in context areas. For details, see **Table 11-108**.

Table 11-108 SQL statements for defining a cursor

Function	SQL Statement
Creating a cursor	**CURSOR**
Moving a cursor	**MOVE**
Fetching data from a cursor	**FETCH**
Closing a cursor	**CLOSE**

Defining an Aggregate Function

Table 11-109 SQL statements for defining an aggregate function

Function	SQL Statement
Creating an aggregate function	**CREATE AGGREGATE**
Modifying an aggregate function	**ALTER AGGREGATE**
Deleting an aggregate function	**DROP AGGREGATE**

Defining Data Type Conversion

Table 11-110 SQL statements for defining a data type

Function	SQL Statement
Creating user-defined data type conversion	**CREATE CAST**
Deleting user-defined data type conversion	**DROP CAST**

Defining a Plug-in Extension

Table 11-111 SQL statements for defining a plug-in extension

Function	SQL Statement
Creating a plug-in extension	**CREATE EXTENSION**
Modifying a plug-in extension	**ALTER EXTENSION**

Function	SQL Statement
Deleting a plug-in extension	**DROP EXTENSION**

Defining an Operator

Table 11-112 SQL statements for defining an operator

Function	SQL Statement
Creating an operator	**CREATE OPERATOR**
Modifying an operator	**ALTER OPERATOR**
Deleting an operator	**DROP OPERATOR**

Defining a Data Type

Table 11-113 SQL statements for defining a data type

Function	SQL Statement
Creating a data type	**CREATE TYPE**
Modifying a data type	**ALTER TYPE**
Deleting a data type	**DROP TYPE**

11.12 DML Syntax Overview

Data manipulation language (DML) is used to perform operations on data in database tables, such as inserting, updating, querying, or deleting data.

Inserting Data

Inserting data refers to adding one or multiple records to a database table. For details, see **INSERT**.

Updating Data

Updating data refers to modifying one or multiple records in a database table. For details, see **UPDATE**.

Querying Data

The database query statement **SELECT** is used to search required information in a database. For details, see **SELECT**.

Deleting Data

GaussDB provides two statements for deleting data from database tables. To delete data meeting specified conditions from a database table, see **DELETE**. To delete all data from a database table, see **TRUNCATE**.

TRUNCATE can quickly delete all data from a database table, which achieves the effect same as that running **DELETE** to delete data without specifying conditions from each table. Deletion efficiency using **TRUNCATE** is faster because **TRUNCATE** does not scan tables. Therefore, **TRUNCATE** is useful in large tables.

Copying Data

GaussDB provides a statement for copying data between tables and files. For details, see **COPY**.

Locking a Table

GaussDB provides multiple lock modes to control concurrent accesses to table data. For details, see **LOCK**.

Calling a Function

GaussDB provides three statements for calling functions. These statements are the same in the syntax structure. For details, see **CALL**.

Session Management

A session is a connection established between the user and the database. **Table 11-114** lists the related SQL statements.

Table 11-114 SQL statements related to sessions

Function	SQL Statement
Altering a session	**ALTER SESSION**
Killing a session	**ALTER SYSTEM KILL SESSION**

11.13 DCL Syntax Overview

Data control language (DCL) is used to create users and roles and set or modify database users or role rights.

Defining a Role

A role is used to manage permissions. For database security, management and operation permissions can be granted to different roles. For details about related SQL statements, see **Table 11-115**.

Table 11-115 SQL statements for defining a role

Description	SQL Statement
Creating a role	**CREATE ROLE**
Altering role attributes	**ALTER ROLE**
Dropping a role	**DROP ROLE**

Defining a User

A user is used to log in to a database. Different permissions can be granted to users for managing data accesses and operations of the users. For details about related SQL statements, see **Table 11-116**.

Table 11-116 SQL statements for defining a user

Description	SQL Statement
Creating a User	**CREATE USER**
Altering user attributes	**ALTER USER**
Dropping a user	**DROP USER**

Granting Rights

GaussDB provides a statement for granting rights to data objects and roles. For details, see **GRANT**.

Revoking Rights

GaussDB provides a statement for revoking rights. For details, see **REVOKE**.

Setting Default Rights

GaussDB allows users to set rights for objects that will be created. For details, see .**ALTER DEFAULT PRIVILEGES**

Shutting Down The Current Node

GaussDB allows users to run the **shutdown** command to shut down the current database node. For details, see **SHUTDOWN**.

11.14 SQL Syntax

11.14.1 ABORT

Function

ABORT rolls back the current transaction and cancels the changes in the transaction.

This command is equivalent to **ROLLBACK**, and is present only for historical reasons. Now **ROLLBACK** is recommended.

Precautions

ABORT has no impact outside a transaction, but will throw a NOTICE message.

Syntax

```
ABORT [ WORK | TRANSACTION ] ;
```

Parameter Description

WORK | TRANSACTION

Specifies an optional keyword, which has no effect except increasing readability.

Examples

```
-- Create the customer_demographics_t1 table.
openGauss=# CREATE TABLE customer_demographics_t1
(
    CD_DEMO_SK                INTEGER               NOT NULL,
    CD_GENDER                 CHAR(1)                       ,
    CD_MARITAL_STATUS         CHAR(1)                       ,
    CD_EDUCATION_STATUS       CHAR(20)                      ,
    CD_PURCHASE_ESTIMATE      INTEGER                       ,
    CD_CREDIT_RATING          CHAR(10)                      ,
    CD_DEP_COUNT              INTEGER                       ,
    CD_DEP_EMPLOYED_COUNT     INTEGER                       ,
    CD_DEP_COLLEGE_COUNT      INTEGER
)
WITH (ORIENTATION = COLUMN,COMPRESSION=MIDDLE)
;

-- Insert data.
openGauss=# INSERT INTO customer_demographics_t1 VALUES(1920801,'M', 'U', 'DOCTOR DEGREE', 200,
'GOOD', 1, 0,0);

-- Start a transaction.
openGauss=# START TRANSACTION;

-- Update the column.
openGauss=# UPDATE customer_demographics_t1 SET cd_education_status= 'Unknown';

-- Abort the transaction. All updates are rolled back.
openGauss=# ABORT;

-- Query data.
openGauss=# SELECT * FROM customer_demographics_t1 WHERE cd_demo_sk = 1920801;
cd_demo_sk | cd_gender | cd_marital_status | cd_education_status  | cd_purchase_estimate | cd_credit_rating
| cd_dep_count | cd_dep_employed_count | cd_dep_college_count
------------+-----------+-------------------+----------------------+----------------------+------------------
+--------------+-----------------------+----------------------
    1920801 | M         | U                 | DOCTOR DEGREE        |                  200 | GOOD             |            1
|                    0 |                    0
```

```
(1 row)

-- Delete the table.
openGauss=# DROP TABLE customer_demographics_t1;
```

Helpful Links

SET TRANSACTION, **COMMIT | END**, and **ROLLBACK**

11.14.2 ALTER AGGREGATE

Function

ALTER AGGREGATE modifies the definition of an aggregate function.

Precautions

To use **ALTER AGGREGATE**, you must be the owner of the aggregate function. To change the schema of an aggregate function, you must have the **CREATE** permission on the new schema. To change the owner, you must be a direct or indirect member of the new role, and the role must have the **CREATE** permission on the aggregate function's schema. (This restricts the owner from doing anything except for deleting and recreating aggregate functions. However, a user with the SYSADMIN permission can change the ownership of an aggregate function in any way.)

Syntax

```
ALTER AGGREGATE name ( argtype [ , ... ] ) RENAME TO new_name
ALTER AGGREGATE name ( argtype [ , ... ] ) OWNER TO new_owner
ALTER AGGREGATE name ( argtype [ , ... ] ) SET SCHEMA new_schema
```

Parameter Description

- **name**

  Name (optionally schema-qualified) of an existing aggregate function.
- **argtype**

  Input data type of the aggregate function. To reference a zero-parameter aggregate function, you can write an asterisk (*) instead of a list of input data types.
- **new_name**

  New name of the aggregate function.
- **new_owner**

  New owner of the aggregate function.
- **new_schema**

  New schema of the aggregate function.

Examples

Rename the aggregate function **myavg** that accepts integer-type parameters to **my_average**.

```
ALTER AGGREGATE myavg(integer) RENAME TO my_average;
```

Change the owner of the aggregate function **myavg** that accepts integer-type parameters to **joe**.

```
ALTER AGGREGATE myavg(integer) OWNER TO joe;
```

Move the aggregate function **myavg** that accepts integer-type parameters to **myschema**.

```
ALTER AGGREGATE myavg(integer) SET SCHEMA myschema;
```

Compatibility

The SQL standard does not contain the **ALTER AGGREGATE** statement.

11.14.3 ALTER AUDIT POLICY

Function

ALTER AUDIT POLICY modifies the unified audit policy.

Precautions

- Only user **poladmin**, user **sysadmin**, or the initial user can perform this operation.
- The unified audit policy takes effect only after **enable_security_policy** is set to **on**. For details, see "Database Configuration > Database Security Management Policies > Unified Auditing" in *Security Hardening Guide*.

Syntax

```
ALTER AUDIT POLICY [ IF EXISTS ] policy_name { ADD | REMOVE } { [ privilege_audit_clause ]
[ access_audit_clause ] };
ALTER AUDIT POLICY [ IF EXISTS ] policy_name MODIFY ( filter_group_clause );
ALTER AUDIT POLICY [ IF EXISTS ] policy_name DROP FILTER;
ALTER AUDIT POLICY [ IF EXISTS ] policy_name COMMENTS policy_comments;
ALTER AUDIT POLICY [ IF EXISTS ] policy_name { ENABLE | DISABLE };
```

- privilege_audit_clause
  ```
  PRIVILEGES { DDL | ALL }
  ```
- access_audit_clause
  ```
  ACCESS { DML | ALL }
  ```
- filter_group_clause
  ```
  FILTER ON { ( FILTER_TYPE ( filter_value [, ... ] ) ) [, ... ] }
  ```

Parameter Description

- **policy_name**

  Specifies the audit policy name, which must be unique.

  Value range: a string. It must comply with the identifier naming convention.
- **DDL**

  Specifies the operations that are audited in the database: **CREATE**, **ALTER**, **DROP**, **ANALYZE**, **COMMENT**, **GRANT**, **REVOKE**, **SET**, **SHOW**, **LOGIN_ANY**, **LOGIN_FAILURE**, **LOGIN_SUCCESS**, and **LOGOUT**.
- **ALL**

  Specifies all operations supported by the specified DDL statements in the database.

- **DML**

  Specifies the operations that are audited in the database: **SELECT**, **COPY**, **DEALLOCATE**, **DELETE**, **EXECUTE**, **INSERT**, **PREPARE**, **REINDEX**, **TRUNCATE**, and **UPDATE**.
- **FILTER_TYPE**

  Specifies the types of information to be filtered by the policy: **IP**, **ROLES**, and **APP**.
- **filter_value**

  Specifies the detailed information to be filtered.
- **policy_comments**

  Records description information of the audit policy.
- **ENABLE|DISABLE**

  Enables or disables the unified audit policy. If **ENABLE|DISABLE** is not specified, **ENABLE** is used by default.

Examples

See **Examples** in **CREATE AUDIT POLICY**.

Helpful Links

CREATE AUDIT POLICY and **DROP AUDIT POLICY**

11.14.4 ALTER DATABASE

Function

ALTER DATABASE modifies a database, including its name, owner, connection limitation, and object isolation.

Precautions

- Only the database owner or a user granted with the ALTER permission can run the **ALTER DATABASE** command. The system administrator has this permission by default. The following is permission constraints depending on attributes to be modified:
  - To modify the database name, you must have the **CREATEDB** permission.
  - To modify a database owner, you must be a database owner or system administrator and a member of the new owner role, with the **CREATEDB** permission.
  - To modify the default tablespace of a database, a user must have the permission to create a tablespace. This statement physically migrates tables and indexes in a default tablespace to a new tablespace. Note that tables and indexes outside the default tablespace are not affected.
- You are not allowed to rename a database in use. To rename it, connect to another database.

Syntax

- Modify the maximum number of connections to the database.
```
ALTER DATABASE database_name
    [ [ WITH ] CONNECTION LIMIT connlimit ];
```
- Rename the database.
```
ALTER DATABASE database_name
    RENAME TO new_name;
```
- Change the database owner.
```
ALTER DATABASE database_name
    OWNER TO new_owner;
```
- Change the default tablespace of the database.
```
ALTER DATABASE database_name
    SET TABLESPACE new_tablespace;
```
- Modify the session parameter value of the database.
```
ALTER DATABASE database_name
    SET configuration_parameter { { TO | = } { value | DEFAULT } | FROM CURRENT };
```
- Reset the database configuration parameter.
```
ALTER DATABASE database_name RESET
    { configuration_parameter | ALL };
```
- Modify the object isolation attribute of the database.
```
ALTER DATABASE database_name [ WITH ] { ENABLE | DISABLE } PRIVATE OBJECT;
```

NOTE

- To modify the object isolation attribute of a database, the database must be connected. Otherwise, the modification will fail.
- For a new database, the object isolation attribute is disabled by default. After this attribute is enabled, common users can view only the objects (such as tables, functions, views, and columns) that they have the permission to access. This attribute does not take effect for administrators. After this attribute is enabled, administrators can still view all database objects.

Parameter Description

- **database_name**

  Specifies the name of the database whose attributes are to be modified.

  Value range: a string. It must comply with the naming convention rule.
- **connlimit**

  Specifies the maximum number of concurrent connections that can be made to this database (excluding administrators' connections).

  Value range: The value must be an integer, preferably from 1 to 50. The default value **-1** indicates that there is no restriction on the number of concurrent connections.
- **new_name**

  Specifies the new name of a database.

  Value range: a string. It must comply with the naming convention rule.
- **new_owner**

  Specifies the new owner of a database.

  Value range: a string. It must be a valid username.
- **new_tablespace**

  Specifies the new default tablespace of a database. The tablespace exists in the database. The default tablespace is **pg_default**.

Value range: a string. It must be a valid tablespace name.

- **configuration_parameter**

  value

  Sets a specified database session parameter to a specified value. If the value is **DEFAULT** or **RESET**, the default setting is used in the new session. **OFF** closes the setting.

  Value range: a string

  - DEFAULT
  - OFF
  - RESET
- **FROM CURRENT**

  Sets the value of the database based on the current connected session.
- **RESET configuration_parameter**

  Resets the specified database session parameter.
- **RESET ALL**

  Resets all database session parameters.

NOTE

- Modify the default tablespace of a database by moving the table or index in the old tablespace into the new tablespace. This operation does not affect the tables or indexes in other non-default tablespaces.
- The modified database session parameter values will take effect in the next session.

Examples

See **Examples** in **CREATE DATABASE**.

Helpful Links

CREATE DATABASE and **DROP DATABASE**

11.14.5 ALTER DATA SOURCE

Function

ALTER DATA SOURCE modifies the attributes and content of the data source.

The attributes include the name and owner. The content includes the type, version, and connection options.

Precautions

- Only the initial user, system administrator, and owner have the permission to modify data sources.
- To change the owner, the new owner must be the initial user or a system administrator.
- If the **password** option is displayed, ensure that the **datasource.key.cipher** and **datasource.key.rand** files exist in the *$GAUSSHOME*/**bin** directory of each node in the database. If the two files do not exist, use the **gs_guc** tool to

generate them and use the **gs_ssh** tool to release them to the *$GAUSSHOME*/**bin** directory on each node.

Syntax

```
ALTER DATA SOURCE src_name
    [TYPE  'type_str']
    [VERSION  {'version_str' | NULL}]
    [OPTIONS ( {[ ADD | SET | DROP ] optname ['optvalue']} [, ...] )];
ALTER DATA SOURCE src_name RENAME TO src_new_name;
ALTER DATA SOURCE src_name OWNER TO new_owner;
```

Parameter Description

- **src_name**

  Specifies the data source name to be modified.

  Value range: a string. It must comply with the identifier naming convention.

- **TYPE**

  Changes the original **TYPE** value of the data source to the specified value.

  Value range: an empty string or a non-empty string

- **VERSION**

  Changes the original **VERSION** value of the data source to the specified value.

  Value range: an empty string, a non-empty string, or null

- **OPTIONS**

  Specifies the column to be added, modified, or deleted. The value of **optname** should be unique. Comply with the following rules to set this parameter:

  To add a column, you can omit **ADD** and simply specify the column name, which cannot be an existing column name.

  To modify a column, specify **SET** and an existing column name.

  To delete a column, specify **DROP** and an existing column name. Do not set **optvalue**.

- **src_new_name**

  Specifies the new data source name.

  Value range: a string. It must comply with the naming convention rule.

- **new_user**

  Specifies the new owner of an object.

  Value range: a string. It must be a valid username.

Examples

```
-- Create an empty data source.
openGauss=# CREATE DATA SOURCE ds_test1;

-- Rename the data source.
openGauss=# ALTER DATA SOURCE ds_test1 RENAME TO ds_test;

-- Change the owner.
openGauss=# CREATE USER user_test1 IDENTIFIED BY 'Gs@123456';
openGauss=# ALTER USER user_test1 WITH SYSADMIN;
openGauss=# ALTER DATA SOURCE ds_test OWNER TO user_test1;

-- Modify TYPE and VERSION.
```

```
openGauss=# ALTER DATA SOURCE ds_test TYPE 'MPPDB_TYPE' VERSION 'XXX';

-- Add a column.
openGauss=# ALTER DATA SOURCE ds_test OPTIONS (add dsn 'gaussdb', username 'test_user');

-- Modify a column.
openGauss=# ALTER DATA SOURCE ds_test OPTIONS (set dsn 'unknown');

-- Delete a column.
openGauss=# ALTER DATA SOURCE ds_test OPTIONS (drop username);

-- Delete the data source and user objects.
openGauss=# DROP DATA SOURCE ds_test;
openGauss=# DROP USER user_test1;
```

Helpful Links

CREATE DATA SOURCE and **DROP DATA SOURCE**

11.14.6 ALTER DEFAULT PRIVILEGES

Function

ALTER DEFAULT PRIVILEGES allows you to set the permissions that will be applied to objects created in the future. (It does not affect permissions granted to existing objects.)

Precautions

Currently, you can change only the permissions for tables (including views), sequences, functions, types, CMKs of encrypted databases, and CEKs.

Syntax

```
ALTER DEFAULT PRIVILEGES
    [ FOR { ROLE | USER } target_role [, ...] ]
    [ IN SCHEMA schema_name [, ...] ]
    abbreviated_grant_or_revoke;
```

- **abbreviated_grant_or_revoke** grants or revokes permissions on some objects.
```
grant_on_tables_clause
  | grant_on_sequences_clause
  | grant_on_functions_clause
  | grant_on_types_clause
  | grant_on_client_master_keys_clause
  | grant_on_column_encryption_keys_clause
  | revoke_on_tables_clause
  | revoke_on_sequences_clause
  | revoke_on_functions_clause
  | revoke_on_types_clause
  | revoke_on_client_master_keys_clause
  | revoke_on_column_encryption_keys_clause
```
- **grant_on_tables_clause** grants permissions on tables.
```
GRANT { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | ALTER | DROP |
COMMENT | INDEX | VACUUM }
    [, ...] | ALL [ PRIVILEGES ] }
    ON TABLES
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ]
```
- **grant_on_sequences_clause** grants permissions on sequences.
```
GRANT { { SELECT | UPDATE | USAGE | ALTER | DROP | COMMENT }
    [, ...] | ALL [ PRIVILEGES ] }
    ON SEQUENCES
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ]
```

- **grant_on_functions_clause** grants permissions on functions.
```
GRANT { { EXECUTE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON FUNCTIONS
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ]
```
- **grant_on_types_clause** grants permissions on types.
```
GRANT { { USAGE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON TYPES
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ]
```
- **grant_on_client_master_keys_clause** grants permissions on CMKs.
```
GRANT { { USAGE  | DROP } [, ...] | ALL [ PRIVILEGES ] }
    ON CLIENT_MASTER_KEYS
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ]
```
- **grant_on_column_encryption_keys_clause** grants permissions on CEKs.
```
GRANT { { USAGE | DROP } [, ...] | ALL [ PRIVILEGES ] }
    ON COLUMN_ENCRYPTION_KEYS
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ]
```
- **revoke_on_tables_clause** revokes permissions on tables.
```
REVOKE [ GRANT OPTION FOR ]
    { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | ALTER | DROP | COMMENT |
INDEX | VACUUM }
    [, ...] | ALL [ PRIVILEGES ] }
    ON TABLES
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT | CASCADE CONSTRAINTS ]
```
- **revoke_on_sequences_clause** revokes permissions on sequences.
```
REVOKE [ GRANT OPTION FOR ]
    { { SELECT | UPDATE | USAGE | ALTER | DROP | COMMENT }
    [, ...] | ALL [ PRIVILEGES ] }
    ON SEQUENCES
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT | CASCADE CONSTRAINTS ]
```
- **revoke_on_functions_clause** revokes permissions on functions.
```
REVOKE [ GRANT OPTION FOR ]
    { {EXECUTE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON FUNCTIONS
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT | CASCADE CONSTRAINTS ]
```
- **revoke_on_types_clause** revokes permissions on types.
```
REVOKE [ GRANT OPTION FOR ]
    { { USAGE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON TYPES
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT | CASCADE CONSTRAINTS ]
```
- **revoke_on_client_master_keys_clause** revokes permissions on CMKs.
```
REVOKE [ GRANT OPTION FOR ]
    { { USAGE | DROP } [, ...] | ALL [ PRIVILEGES ] }
    ON CLIENT_MASTER_KEYS
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT | CASCADE CONSTRAINTS ]
```
- **revoke_on_column_encryption_keys_clause** revokes permissions on CEKs.
```
REVOKE [ GRANT OPTION FOR ]
    { { USAGE | DROP } [, ...] | ALL [ PRIVILEGES ] }
    ON COLUMN_ENCRYPTION_KEYS
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT | CASCADE CONSTRAINTS ]
```

Parameter Description

- **target_role**

  Specifies the name of an existing role. If **FOR ROLE/USER** is omitted, the current role is assumed.

  Value range: an existing role name
- **schema_name**

  Specifies the name of an existing schema.

  target_role must have the **CREATE** permission for **schema_name**.

  Value range: an existing schema name
- **role_name**

  Specifies the name of an existing role to grant or revoke permissions for.

  Value range: an existing role name

NOTICE

To drop a role for which the default permissions have been granted, reverse the changes in its default permissions or use **DROP OWNED BY** to get rid of the default permission entry for the role.

Examples

```
-- Grant the SELECT permission on all the tables (and views) in tpcds to every user.
openGauss=# ALTER DEFAULT PRIVILEGES IN SCHEMA tpcds GRANT SELECT ON TABLES TO PUBLIC;

-- Create a common user jack.
openGauss=# CREATE USER jack PASSWORD 'xxxxxxxxx';

-- Grant the INSERT permission on all the tables in tpcds to the user jack.
openGauss=# ALTER DEFAULT PRIVILEGES IN SCHEMA tpcds GRANT INSERT ON TABLES TO jack;

-- Revoke the preceding permissions.
openGauss=# ALTER DEFAULT PRIVILEGES IN SCHEMA tpcds REVOKE SELECT ON TABLES FROM PUBLIC;
openGauss=# ALTER DEFAULT PRIVILEGES IN SCHEMA tpcds REVOKE INSERT ON TABLES FROM jack;

-- Delete user jack.
openGauss=# DROP USER jack;
```

Helpful Links

GRANT and **REVOKE**

11.14.7 ALTER DIRECTORY

Function

ALTER DIRECTORY modifies a directory.

Precautions

- Currently, only the directory owner can be changed.
- When **enable_access_server_directory** is set to **off**, only the initial user is allowed to change the directory owner. When

enable_access_server_directory is set to **on**, users with the **SYSADMIN** permission and the directory object owner can change the directory object owner, and the user who changes the owner is required to be a member of the new owner.

Syntax

```
ALTER DIRECTORY directory_name
    OWNER TO new_owner;
```

Parameter Description

directory_name

Specifies the name of a directory to be modified. The value must be an existing directory name.

Examples

```
-- Create a directory.
openGauss=# CREATE OR REPLACE DIRECTORY  dir  as '/tmp/';

-- Change the owner of the directory.
openGauss=# ALTER DIRECTORY dir OWNER TO system;

-- Delete a directory.
openGauss=# DROP DIRECTORY dir;
```

Helpful Links

CREATE DIRECTORY and **DROP DIRECTORY**

11.14.8 ALTER EXTENSION

Function

ALTER EXTENSION modifies the plug-in extension.

Precautions

ALTER EXTENSION modifies the definition of an installed extension. Methods are as follows:

- UPDATE

  The extension is updated to a new version. The extension must be applicable to an update script (or a series of scripts) so that the current installation version can be modified to a required version.
- SET SCHEMA

  The extended object is moved to another schema. This extension must be relocatable to make the command successful.
- ADD member_object

  An existing object is added to the extension. This is mainly useful for extension update scripts. This object is then treated as a member of the extension. Obviously, the object can only be canceled by canceling the extension.

- DROP member_object

  A member object is removed from the extension. This method is useful in extension update scripts. The object is not canceled, but is simply separated from the extension.

  You must have an extension before using **ALTER EXTENSION**. You must have the permission on adding or deleting an object before using the ADD or DROP statement.

Syntax

```
ALTER EXTENSION name UPDATE [ TO new_version ]
ALTER EXTENSION name SET SCHEMA new_schema
ALTER EXTENSION name ADD member_object
ALTER EXTENSION name DROP member_object

where member_object is:

  AGGREGATE agg_name (agg_type [, ...] ) |
  CAST (source_type AS target_type) |
  COLLATION object_name |
  CONVERSION object_name |
  DOMAIN object_name |
  EVENT TRIGGER object_name |
  FOREIGN DATA WRAPPER object_name |
  FOREIGN TABLE object_name |
  FUNCTION function_name ( [ [ argname ] [ argmode ] argtype [, ...] ] ) |
  MATERIALIZED VIEW object_name |
  OPERATOR operator_name (left_type, right_type) |
  OPERATOR CLASS object_name USING index_method |
  OPERATOR FAMILY object_name USING index_method |
  [ PROCEDURAL ] LANGUAGE object_name |
  SCHEMA object_name |
  SEQUENCE object_name |
  SERVER object_name |
  TABLE object_name |
  TEXT SEARCH CONFIGURATION object_name |
  TEXT SEARCH DICTIONARY object_name |
  TEXT SEARCH PARSER object_name |
  TEXT SEARCH TEMPLATE object_name |
  TYPE object_name |
  VIEW object_name
```

Parameter Description

- **name**

  Name of an installed extension.

- **new_version**

  New version of the extension, which can be overridden by identifiers and string literals. If a new version of the extension is not specified, ALTER EXTENSION UPDATE updates to the default version shown in the extension's control file.

- **new_schema**

  New schema of the extension.

- **object_name**

  agg_name

  function_name

  operator_name

Names of objects that are added or removed from the extension, including names of tables, aggregations, domains, external linked lists, functions, operators, operator classes, operator families, sequences, text search objects, types, and views that can be schema-qualified.

- **agg_type**

  Input data type of the aggregate function. To reference a zero-parameter aggregate function, use ***** to replace the input data type list.

- **source_type**

  Name of the source data type to be forcibly converted.

- **target_type**

  Name of the target data type to be forcibly converted.

- **argmode**

  Model of the function parameter. The value can be **IN**, **OUT**, **INOUT**, or **VARIADIC**. The default value is **IN**. **ALTER EXTENSION** does not relate to the **OUT** parameter, because you only need to enter parameters to confirm the consistency of functions. Therefore, the **IN**, **INOUT**, and **VARIADIC** parameters are enough.

- **argname**

  Name of a function parameter. ALTER EXTENSION does not relate to the parameter name. Only the parameter data type is required to confirm the consistency of the function.

- **argtype**

  Data type (optionally schema-qualified) of a function parameter.

- **left_type**

  right_type

  Data type (optionally schema-qualified) of an operator parameter. **NONE** is written for a missing parameter of a prefix or suffix operator.

Examples

Update the hstore extension to version 2.0.

```
ALTER EXTENSION hstore UPDATE TO '2.0';
```

Run the following command to update the hstore extension mode to utils.

```
ALTER EXTENSION hstore SET SCHEMA utils;
```

Add an existing function for hstore extension.

```
ALTER EXTENSION hstore ADD FUNCTION populate_record(anyelement, hstore);
```

11.14.9 ALTER FOREIGN TABLE

Function

ALTER FOREIGN TABLE modifies a foreign table.

Precautions

When multi-layer quotation marks are used for sensitive columns (such as **password** and **secret_access_key**) in OPTIONS, the semantics is different from that in the scenario where quotation marks are not used. Therefore, sensitive columns are not identified for anonymization.

Syntax

```
ALTER FOREIGN TABLE [ IF EXISTS ]  table_name
    OPTIONS ( {[ ADD | SET | DROP ] option ['value']}[, ... ]);

ALTER FOREIGN TABLE [ IF EXISTS ] table_name
    ALTER column_name OPTIONS;
```

Parameter Description

- **table_name**

  Specifies the name of an existing foreign table to be modified.

  Value range: an existing foreign table name.

- **option**

  Specifies the option of a foreign table or foreign table column to be modified. **ADD**, **SET**, and **DROP** are operations to be performed. If no operation is set explicitly, the default value **ADD** is used. The option name must be unique (although table options and table column options can share the same name). The name and value of the option are also validated by a class library of a foreign data wrapper.

  – Options supported by **file_fdw** are as follows:

    ■ filename

      File to be read. This parameter is mandatory and must be an absolute path.

    ■ format

      File format of the remote server, which is the same as the **FORMAT** option in the **COPY** statement. The value can be **text**, **csv**, **binary**, or **fixed**.

    ■ header

      Specifies whether a specified file has a header, which is the same as the **HEADER** option of the **COPY** statement.

      ○ delimiter

        File delimiter, which is the same as the **DELIMITER** option of the **COPY** statement.

      ○ quote

        Quote character of a file, which is the same as the **QUOTE** option of the **COPY** statement.

      ○ escape

        Escape character of a file, which is the same as the **ESCAPE** option of the **COPY** statement.

      ○ null

  Null string of a file, which is the same as the **NULL** option of the **COPY** statement.

  - encoding

    Encoding of a file, which is the same as the **ENCODING** option of the **COPY** statement.

  - force_not_null

    This is a Boolean option. If it is true, the value of the declared field cannot be an empty string. This option is the same as the **FORCE_NOT_NULL** option of the **COPY** statement.

- **value**

  Specifies the new value of **option**.

Helpful Links

CREATE FOREIGN TABLE and **DROP FOREIGN TABLE**

11.14.10 ALTER FUNCTION

Function

ALTER FUNCTION modifies the attributes of a customized function.

Precautions

Only the function owner or a user granted with the ALTER permission can run the **ALTER FUNCTION** command. The system administrator has this permission by default. The following is permission constraints depending on attributes to be modified:

- If a function involves operations on temporary tables, **ALTER FUNCTION** cannot be used.
- To modify the owner or schema of a function, you must be a function owner or system administrator and a member of the new owner role.
- Only the system administrator and initial user can change the schema of a function to **public**.

Syntax

- Modify the additional parameters of the customized function.

```
ALTER FUNCTION function_name ( [ { [ argname ] [ argmode ] argtype} [, ...] ] )
    action [ ... ] [ RESTRICT ];
```

  The syntax of the **action** clause is as follows:

```
{CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT}
 | {IMMUTABLE | STABLE | VOLATILE}
 | {SHIPPABLE | NOT SHIPPABLE}
 | {NOT FENCED | FENCED}
 | [ NOT ] LEAKPROOF
 | { [ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER }
 | AUTHID { DEFINER | CURRENT_USER }
 | COST execution_cost
 | ROWS result_rows
 | SET configuration_parameter { { TO | = } { value | DEFAULT }| FROM CURRENT}
 | RESET {configuration_parameter | ALL}
```

- Rename the customized function.
  ```
  ALTER FUNCTION funname ( [ { [ argname ] [ argmode ] argtype} [, ...] ] )
     RENAME TO new_name;
  ```
- Change the owner of the customized function.
  ```
  ALTER FUNCTION funname ( [ { [ argname ] [ argmode ] argtype} [, ...] ] )
     OWNER TO new_owner;
  ```
- Modify the schema of the customized function.
  ```
  ALTER FUNCTION funname ( [ { [ argname ] [ argmode ] argtype} [, ...] ] )
     SET SCHEMA new_schema;
  ```

Parameter Description

- **function_name**

  Specifies the name of the function to be modified.

  Value range: an existing function name
- **argmode**

  Specifies whether a parameter is an input or output parameter.

  Value range: **IN**, **OUT**, **INOUT**, and **VARIADIC**
- **argname**

  Parameter name.

  Value range: a string. It must comply with the identifier naming convention.
- **argtype**

  Specifies the data type of a function parameter.
- **CALLED ON NULL INPUT**

  Declares that some parameters of the function can be invoked in normal mode if the parameter values are null. Omitting this parameter is the same as specifying it.
- **RETURNS NULL ON NULL INPUT**

  STRICT

  Specifies that the function always returns null whenever any of its parameters is null. If **STRICT** is specified, the function will not be executed when there are null parameters; instead a null result is assumed automatically.

  RETURNS NULL ON NULL INPUT and **STRICT** have the same functions.
- **IMMUTABLE**

  Specifies that the function always returns the same result if the parameter values are the same.
- **STABLE**

  Specifies that the function cannot modify the database, and that within a single table scan it will consistently return the same result for the same parameter value, but its result varies by SQL statements.
- **VOLATILE**

  Specifies that the function value can change in a single table scan and no optimization is performed.
- **LEAKPROOF**

  Specifies that the function has no side effect and the parameter contains only the return value. **LEAKPROOF** can be set only by the system administrator.

- **EXTERNAL**

  (Optional) The purpose is to be compatible with SQL. This feature applies to all functions, not only external functions.
- **SECURITY INVOKER**

  AUTHID CURRENT_USER

  Specifies that the function will be executed with the permissions of the user who invokes it. Omitting this parameter is the same as specifying it.

  SECURITY INVOKER and **AUTHID CURRENT_USER** have the same functions.
- **SECURITY DEFINER**

  AUTHID DEFINER

  Specifies that the function will be executed with the permissions of the user who created it.

  AUTHID DEFINER and **SECURITY DEFINER** have the same function.
- **COST execution_cost**

  Estimates the execution cost of a function.

  The unit of **execution_cost** is **cpu_operator_cost**.

  Value range: a positive integer
- **ROWS result_rows**

  Estimates the number of rows returned by the function. This is only allowed when the function is declared to return a set.

  Value range: a positive number. The default value is **1000**.
- **configuration_parameter**
  - **value**

    Sets a specified database session parameter to a specified value. If the value is **DEFAULT** or **RESET**, the default setting is used in the new session. **OFF** closes the setting.

    Value range: a string
    - DEFAULT
    - OFF
    - RESET

    Specifies the default value.
  - **from current**

    Uses the value of **configuration_parameter** of the current session.
- **new_name**

  Specifies the new name of a function. To change the schema of a function, you must have the **CREATE** permission on the new schema.

  Value range: a string. It must comply with the identifier naming convention.
- **new_owner**

  Specifies the new owner of a function. To change the owner of a function, the new owner must have the **CREATE** permission on the schema to which the function belongs.

  Value range: an existing user role

- **new_schema**

  Specifies the new schema of a function.

  Value range: an existing schema

Examples

See **Examples** in **CREATE FUNCTION**.

Helpful Links

CREATE FUNCTION and **DROP FUNCTION**

11.14.11 ALTER GLOBAL CONFIGURATION

Function

ALTER GLOBAL CONFIGURATION adds and modifies the **gs_global_config** system catalog and adds the value of **key-value**.

Precautions

- Only the initial database user can run this command.
- The keyword cannot be changed to **weak_password**.

Syntax

```
ALTER GLOBAL CONFIGURATION with(paraname=value, paraname=value...);
```

Parameter Description

The parameter name and value are of the text type.

11.14.12 ALTER GROUP

Function

ALTER GROUP modifies the attributes of a user group.

Precautions

ALTER GROUP is an alias for **ALTER ROLE**, and it is not a standard SQL syntax and not recommended. Users can use **ALTER ROLE** directly.

Syntax

- Add users to a group.
  ```
  ALTER GROUP group_name
      ADD USER user_name [, ... ];
  ```
- Remove users from a group.
  ```
  ALTER GROUP group_name
      DROP USER user_name [, ... ];
  ```
- Change the name of the group.

```
ALTER GROUP group_name
    RENAME TO new_name;
```

Parameter Description

See **Parameter Description** in **ALTER ROLE**.

Examples

```
-- Add users to a group.
openGauss=# ALTER GROUP super_users ADD USER lche, jim;

-- Remove users from a group.
openGauss=# ALTER GROUP super_users DROP USER jim;

-- Change the name of the group.
openGauss=# ALTER GROUP super_users RENAME TO normal_users;
```

Helpful Links

ALTER GROUP, **DROP GROUP**, and **ALTER ROLE**

11.14.13 ALTER INDEX

Function

ALTER INDEX modifies the definition of an existing index.

It has the following forms:

- IF EXISTS

  Sends a notice instead of an error if the specified index does not exist.
- RENAME TO

  Changes only the name of the index. The stored data is not affected.
- SET TABLESPACE

  This option changes the index tablespace to the specified tablespace and moves index-related data files to the new tablespace.
- SET ({ STORAGE_PARAMETER = value } [, ...])

  Changes one or more index-method-specific storage parameters of an index. Note that the index content will not be modified immediately by this statement. You may need to use **REINDEX** to recreate the index based on different parameters to achieve the expected effect.
- RESET ({ storage_parameter } [, ...])

  Resets one or more index-method-specific storage parameters of an index to the default value. Similar to the **SET** statement, **REINDEX** may be used to completely update the index.
- [MODIFY PARTITION index_partition_name] UNUSABLE

  Sets the indexes on a table or index partition to be unavailable.
- REBUILD [PARTITION index_partition_name]

  Rebuilds indexes on a table or an index partition.
- RENAME PARTITION

  Renames an index partition.

- MOVE PARTITION

  Modifies the tablespace to which an index partition belongs.

Precautions

Only the index owner or a user who has the INDEX permission on the table where the index resides can run the **ALTER INDEX** command. The system administrator has this permission by default.

Syntax

- Rename a table index.
  ```
  ALTER INDEX [ IF EXISTS ] index_name
      RENAME TO new_name;
  ```
- Change the tablespace to which a table index belongs.
  ```
  ALTER INDEX [ IF EXISTS ] index_name
      SET TABLESPACE tablespace_name;
  ```
- Modify the storage parameter of a table index.
  ```
  ALTER INDEX [ IF EXISTS ] index_name
      SET ( {storage_parameter = value} [, ... ] );
  ```
- Reset the storage parameter of a table index.
  ```
  ALTER INDEX [ IF EXISTS ] index_name
      RESET ( storage_parameter [, ... ] ) ;
  ```
- Set a table index or an index partition to be unavailable.
  ```
  ALTER INDEX [ IF EXISTS ] index_name
      [ MODIFY PARTITION index_partition_name ] UNUSABLE;
  ```

  NOTE

  The syntax cannot be used for column-store tables.
- Rebuild a table index or index partition.
  ```
  ALTER INDEX index_name
      REBUILD [ PARTITION index_partition_name ];
  ```
- Rename an index partition.
  ```
  ALTER INDEX [ IF EXISTS ] index_name
      RENAME PARTITION index_partition_name TO new_index_partition_name;
  ```
- Modify the tablespace to which an index partition belongs.
  ```
  ALTER INDEX [ IF EXISTS ] index_name
      MOVE PARTITION index_partition_name TABLESPACE new_tablespace;
  ```

Parameter Description

- **index_name**

  Specifies the index name to be modified.
- **new_name**

  Specifies the new name of the index.

  Value range: a string. It must comply with the naming convention rule.
- **tablespace_name**

  Specifies the tablespace name.

  Value range: an existing tablespace name
- **storage_parameter**

  Specifies the name of an index-method-specific parameter.

- **value**

  Specifies the new value for an index-method-specific storage parameter. This might be a number or a word depending on the parameter.
- **new_index_partition_name**

  Specifies the new name of the index partition.
- **index_partition_name**

  Specifies the name of an index partition.
- **new_tablespace**

  Specifies a new tablespace.

Examples

See **Examples** in **CREATE INDEX**.

Helpful Links

CREATE INDEX, **DROP INDEX**, and **REINDEX**

11.14.14 ALTER LANGUAGE

This version does not support this syntax.

11.14.15 ALTER LARGE OBJECT

Function

ALTER LARGE OBJECT changes the owner of a large object.

Precautions

Only a system administrator or the owner of the to-be-modified large object can run **ALTER LARGE OBJECT**.

Syntax

```
ALTER LARGE OBJECT large_object_oid
    OWNER TO new_owner;
```

Parameter Description

- **large_object_oid**

  Specifies the OID of the large object to be modified.

  Value range: an existing large object name
- **OWNER TO new_owner**

  Specifies the new owner of an object.

  Value range: an existing username or role name

Examples

None

11.14.16 ALTER MASKING POLICY

Function

ALTER MASKING POLICY modifies masking policies.

Precautions

- Only users with the **poladmin** or **sysadmin** permission, or the initial user can perform this operation.
- The masking policy takes effect only after **enable_security_policy** is set to **on**. For details about how to enable the masking policy, see "Database Configuration > Database Security Management Policies > Dynamic Data Masking" in *Security Hardening Guide*.
- For details about the execution effect and supported data types of preset masking functions, see "Database Security > Dynamic Data Masking" in *Feature Description*.

Syntax

- Modify the policy description.
```
ALTER MASKING POLICY policy_name COMMENTS policy_comments;
```
- Modify the masking method.
```
ALTER MASKING POLICY policy_name [ADD | REMOVE | MODIFY] masking_actions[, ...]*;
The syntax of masking_action.
    masking_function ON LABEL(label_name[, ...]*)
```
- Modify the scenarios where the masking policies take effect.
```
ALTER MASKING POLICY policy_name MODIFY(FILTER ON FILTER_TYPE(filter_value[, ...]*)[, ...]*);
```
- Removes the filters of the masking policies.
```
ALTER MASKING POLICY policy_name DROP FILTER;
```
- Enable or disable the masking policies.
```
ALTER MASKING POLICY policy_name [ENABLE | DISABLE];
```

Parameter Description

- **policy_name**

  Specifies the masking policy name, which must be unique.

  Value range: a string. It must comply with the naming convention.
- **policy_comments**

  Adds or modifies description of masking policies.
- **masking_function**

  Specifies eight preset masking methods or user-defined functions. Schemas are supported.

  maskall is not a preset function. It is hard-coded and cannot be displayed by running **\df**.

  The masking methods during presetting are as follows:
```
maskall | randommasking | creditcardmasking | basicemailmasking | fullemailmasking |
shufflemasking | alldigitsmasking | regexpmasking
```
- **label_name**

  Specifies the resource label name.

- **FILTER_TYPE**

  Specifies the types of information to be filtered by the policies: **IP**, **ROLES**, and **APP**.
- **filter_value**

  Indicates the detailed information to be filtered, such as the IP address, app name, and username.
- **ENABLE|DISABLE**

  Enables or disables the masking policy. If **ENABLE|DISABLE** is not specified, **ENABLE** is used by default.

Examples

```
-- Create users dev_mask and bob_mask.
openGauss=# CREATE USER dev_mask PASSWORD 'dev@1234';
openGauss=# CREATE USER bob_mask PASSWORD 'bob@1234';

-- Create table tb_for_masking.
openGauss=# CREATE TABLE tb_for_masking(col1 text, col2 text, col3 text);

-- Create a resource label for label sensitive column col1.
openGauss=# CREATE RESOURCE LABEL mask_lb1 ADD COLUMN(tb_for_masking.col1);

-- Create a resource label for label sensitive column col2.
openGauss=# CREATE RESOURCE LABEL mask_lb2 ADD COLUMN(tb_for_masking.col2);

-- Create a masking policy for the operation of accessing sensitive column col1.
openGauss=# CREATE MASKING POLICY maskpol1 maskall ON LABEL(mask_lb1);

-- Add description for masking policy maskpol1.
openGauss=# ALTER MASKING POLICY maskpol1 COMMENTS 'masking policy for tb_for_masking.col1';

-- Modify masking policy maskpol1 to add a masking method.
openGauss=# ALTER MASKING POLICY maskpol1 ADD randommasking ON LABEL(mask_lb2);

-- Modify masking policy maskpol1 to remove a masking method.
openGauss=# ALTER MASKING POLICY maskpol1 REMOVE randommasking ON LABEL(mask_lb2);

-- Modify masking policy maskpol1 to modify a masking method.
openGauss=# ALTER MASKING POLICY maskpol1 MODIFY randommasking ON LABEL(mask_lb1);

-- Modify masking policy maskpol1 so that it takes effect only for scenarios where users are dev_mask and
bob_mask, client tools are psql and gsql, and the IP addresses are 10.20.30.40 and 127.0.0.0/24.
openGauss=# ALTER MASKING POLICY maskpol1 MODIFY (FILTER ON ROLES(dev_mask, bob_mask),
APP(psql, gsql), IP('10.20.30.40', '127.0.0.0/24'));

-- Modify masking policy maskpol1 so that it takes effect for all user scenarios.
openGauss=# ALTER MASKING POLICY maskpol1 DROP FILTER;

-- Disable masking policy maskpol1.
openGauss=# ALTER MASKING POLICY maskpol1 DISABLE;
```

Helpful Links

CREATE MASKING POLICY and DROP MASKING POLICY

11.14.17 ALTER MATERIALIZED VIEW

Function

ALTER MATERIALIZED VIEW changes multiple auxiliary attributes of an existing materialized view.

Statements and actions that can be used for **ALTER MATERIALIZED VIEW** are a subset of **ALTER TABLE** and have the same meaning when used for materialized views. For details, see **ALTER TABLE**.

Precautions

- Only the owner of a materialized view or a system administrator has the **ALTER TMATERIALIZED VIEW** permission.
- The materialized view structure cannot be modified.

Syntax

- Change the owner of the materialized view.
```
ALTER MATERIALIZED VIEW [ IF EXISTS ] mv_name
    OWNER TO new_owner;
```
- Modify the column of a materialized view.
```
ALTER MATERIALIZED VIEW [ IF EXISTS ] mv_name
    RENAME [ COLUMN ] column_name TO new_column_name;
```
- Rename a materialized view.
```
ALTER MATERIALIZED VIEW [ IF EXISTS ] mv_name

    RENAME TO new_name;
```

Parameter Description

- **mv_name**

  Specifies the name of an existing materialized view, which can be schema-qualified.

  Value range: a string. It must comply with the naming convention.
- **column_name**

  Specifies the name of a new or existing column.

  Value range: a string. It must comply with the naming convention.
- **new_column_name**

  Specifies the new name of an existing column.
- **new_owner**

  Specifies the user name of the new owner of a materialized view.
- **new_name**

  Specifies the new name of a materialized view.

Examples

```
-- Rename the materialized view foo to bar.
openGauss=# ALTER MATERIALIZED VIEW foo RENAME TO bar;
```

Helpful Links

CREATE MATERIALIZED VIEW, **CREATE INCREMENTAL MATERIALIZED VIEW**, **DROP MATERIALIZED VIEW**, **REFRESH INCREMENTAL MATERIALIZED VIEW**, and **REFRESH MATERIALIZED VIEW**

11.14.18 ALTER OPERATOR

Function

ALTER OPERATOR modifies the definition of an operator.

Precautions

ALTER OPERATOR changes the definition of an operator. Currently, the only function available is to change the owner of the operator.

To use ALTER OPERATOR, you must be the owner of the operator. To modify the owner, you must also be a direct or indirect member of the new owning role, and that member must have CREATE permission on the operator's schema. (These restrictions force the owner to do nothing that cannot be done by deleting and recreating the operator. However, a user with the SYSADMIN permission can modify the ownership of any operator in any way.)

Syntax

```
ALTER OPERATOR name ( { left_type | NONE } , { right_type | NONE } ) OWNER TO new_owner
ALTER OPERATOR name ( { left_type | NONE } , { right_type | NONE } ) SET SCHEMA new_schema
```

Parameter Description

- **name**

  Name of an existing operator.
- **left_type**

  Data type of the left operand for the operator; if there is no left operand, write NONE.
- **right_type**

  Data type of the right operand for the operator; if there is no right operand, write NONE.
- **new_owner**

  New owner of the operator.
- **new_schema**

  New schema name of the operator.

Example

Change a user-defined operator for text a @@ b:

```
ALTER OPERATOR @@ (text, text) OWNER TO joe;
```

Compatibility

The SQL standard does not contain the ALTER OPERATOR statement.

11.14.19 ALTER PUBLICATION

Function

ALTER PUBLICATION alters the attributes of a publication.

Precautions

Only the owner of a publication and the system administrator can execute **ALTER PUBLICATION**. Only the direct or indirect members of the new owner role can change the owner. The new owner must have the **CREATE** permission on the current database. In addition, the new owner published by **FOR ALL TABLES** must be the system administrator. However, the system administrator can change the ownership of a publication while avoiding these restrictions.

Syntax

- Replace the currently published table with a specified table.
  ```
  ALTER PUBLICATION name SET TABLE table_name [, ...]
  ```
- Add one or more tables to a publication.
  ```
  ALTER PUBLICATION name ADD TABLE table_name [, ...]
  ```
- Delete one or more tables from a publication.
  ```
  ALTER PUBLICATION name DROP TABLE table_name [, ...]
  ```
- Change all publication attributes specified in **CREATE PUBLICATION**. Retain previous settings for attributes that are not mentioned.
  ```
  ALTER PUBLICATION name SET ( publication_parameter [= value] [, ... ] )
  ```
- Change the owner of a publication.
  ```
  ALTER PUBLICATION name OWNER TO new_owner
  ```
- Change the name of a publication.
  ```
  ALTER PUBLICATION name RENAME TO new_name
  ```

Parameter Description

- **name**

  Specifies the name of the publication to be modified.
- **table_name**

  Specifies the name of an existing table.
- **SET (publication_parameter [= value] [, ...])**

  Modifies the publication parameters initially set by **CREATE PUBLICATION**.
- **new_owner**

  Specifies the username of the new owner of a publication.
- **new_name**

  Specifies the new name of a publication.

Examples

For details, see **Examples**.

Helpful Links

CREATE PUBLICATION and **DROP PUBLICATION**

11.14.20 ALTER PACKAGE

Function

ALTER PACKAGE alters the attributes of a package.

Precautions

Currently, only users with the **ALTER PACKAGE OWNER** permission can run this command. The system administrator has this permission by default. The restrictions are as follows:

- The current user must be the owner of the package or the system administrator and a member of the new owner role.

Syntax

- Change the owner of a package.

```
ALTER PACKAGE package_name OWNER TO new_owner;
```

Parameter Description

- **package_name**

  Specifies the name of the package to be modified.

  Value range: an existing package name. Only one package can be modified at a time.

- **new_owner**

  Specifies the new owner of a package. To change the owner of a package, the new owner must have the **CREATE** permission on the schema to which the package belongs.

  Value range: an existing user role.

Examples

For details, see **CREATE PACKAGE**.

Helpful Links

CREATE PACKAGE and **DROP PACKAGE**

11.14.21 ALTER PROCEDURE

Function

ALTER PROCEDURE alters the attributes of a customized stored procedure.

Precautions

Only the owner of a stored procedure or a user granted with the **ALTER** permission can run the **ALTER PROCEDURE** command. The system administrator has this permission by default. The following is permission constraints depending on attributes to be modified:

- If a stored procedure involves operations on temporary tables, **ALTER PROCEDURE** cannot be used.
- To modify the owner or schema of a stored procedure, you must be the owner of the stored procedure or system administrator and a member of the new owner role.
- Only the system administrator and initial user can change the schema of a stored procedure to **public**.

Syntax

- Modify the additional parameters of a customized stored procedure.
```
ALTER PROCEDURE procedure_name ( [ { [ argname ] [ argmode ] argtype} [, ...] ] )
    action [ ... ] [ RESTRICT ];
```
  The syntax of the **action** clause is as follows:
```
{CALLED ON NULL INPUT  | STRICT}
 | {IMMUTABLE | STABLE | VOLATILE}
 | {SHIPPABLE | NOT SHIPPABLE}
 | {NOT FENCED | FENCED}
 | [ NOT ] LEAKPROOF
 | { [ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER }
 | AUTHID { DEFINER | CURRENT_USER }
 | COST execution_cost
 | ROWS result_rows
 | SET configuration_parameter { { TO | = } { value | DEFAULT }| FROM CURRENT}
 | RESET {configuration_parameter | ALL}
```
- Modify the name of a customized stored procedure.
```
ALTER PROCEDURE proname ( [ { [ argname ] [ argmode ] argtype} [, ...] ] )
    RENAME TO new_name;
```
- Modify the owner of a customized stored procedure.
```
ALTER PROCEDURE proname ( [ { [ argname ] [ argmode ] argtype} [, ...] ] )
    OWNER TO new_owner;
```
- Modify the schema of a customized stored procedure.
```
ALTER PROCEDURE proname ( [ { [ argname ] [ argmode ] argtype} [, ...] ] )
    SET SCHEMA new_schema;
```

Parameter Description

- **procedure_name**

  Specifies the name of the stored procedure to be modified.

  Value range: an existing stored procedure name
- **argmode**

  Specifies whether a parameter is an input or output parameter.

  Value range: **IN**, **OUT**, **INOUT**, and **VARIADIC**
- **argname**

  Specifies the parameter name.

  Value range: a string. It must comply with the identifier naming convention.
- **argtype**

  Specifies the type of the stored procedure parameter.
- **CALLED ON NULL INPUT**

  Declares that some parameters of the stored procedure can be called in normal mode if the parameter values are null. Omitting this parameter is the same as specifying it.

- **IMMUTABLE**

  Specifies that the stored procedure always returns the same result if the parameter values are the same.

- **STABLE**

  Specifies that the stored procedure cannot modify the database, and that within a single table scan it will consistently return the same result for the same parameter value, but its result varies by SQL statements.

- **VOLATILE**

  Specifies that the stored procedure value can change in a single table scan and no optimization is performed.

- **LEAKPROOF**

  Specifies that the stored procedure has no side effect and the parameter contains only the return value. **LEAKPROOF** can be set only by the system administrator.

- **EXTERNAL**

  (Optional) The purpose is to be compatible with SQL. This feature applies to all functions, not only external functions.

- **SECURITY INVOKER**

  AUTHID CURRENT_USER

  Specifies that the stored procedure will be executed with the permissions of the user who calls it. Omitting this parameter is the same as specifying it.

  SECURITY INVOKER and **AUTHID CURRENT_USER** have the same functions.

- **SECURITY DEFINER**

  AUTHID DEFINER

  Specifies that the stored procedure will be executed with the permissions of the user who created it.

  AUTHID DEFINER and **SECURITY DEFINER** have the same functions.

- **COST execution_cost**

  Estimates the execution cost of the stored procedure.

  The unit of **execution_cost** is **cpu_operator_cost**.

  Value range: a positive integer

- **ROWS result_rows**

  Estimates the number of rows returned by the stored procedure. This is only allowed when the stored procedure is declared to return a set.

  Value range: a positive number. The default value is **1000**.

- **configuration_parameter**
  - **value**

    Sets a specified database session parameter to a specified value. If the value is **DEFAULT** or **RESET**, the default setting is used in the new session. **OFF** disables the setting.

    Value range: a string.

    - DEFAULT

- OFF
- RESET

  Specifies the default value.

– **from current**

  Uses the value of **configuration_parameter** of the current session.

- **new_name**

  Specifies the new name of the stored procedure. To change the schema of a stored procedure, you must have the **CREATE** permission on the new schema.

  Value range: a string. It must comply with the identifier naming convention.

- **new_owner**

  Specifies the new owner of the stored procedure. To change the owner of a stored procedure, the new owner must have the CREATE permission on the schema to which the stored procedure belongs.

  Value range: an existing user role

- **new_schema**

  Specifies the new schema of the stored procedure.

  Value range: an existing schema

Examples

See **Examples** in **CREATE FUNCTION**.

Helpful Links

CREATE PROCEDURE and **DROP PROCEDURE**

11.14.22 ALTER RESOURCE LABEL

Function

ALTER RESOURCE LABEL modifies resource labels.

Precautions

Only users with the **poladmin** or **sysadmin** permission, or the initial user can perform this operation.

Syntax

```
ALTER RESOURCE LABEL label_name (ADD|REMOVE)
  label_item_list[, ...]*;
```

- label_item_list
  ```
  resource_type(resource_path[, ...]*)
  ```
- resource_type
  ```
  TABLE | COLUMN | SCHEMA | VIEW | FUNCTION
  ```

Parameter Description

- **label_name**

  Specifies the resource label name.

  Value range: a string. It must comply with the naming convention.

- **resource_type**

  Specifies the type of database resources to be labeled.

- **resource_path**

  Specifies the path of database resources.

Examples

```
-- Create basic table table_for_label.
openGauss=# CREATE TABLE table_for_label(col1 int, col2 text);

-- Create resource label table_label.
openGauss=# CREATE RESOURCE LABEL table_label ADD COLUMN(table_for_label.col1);

-- Attach resource label table_label to col2.
openGauss=# ALTER RESOURCE LABEL table_label ADD COLUMN(table_for_label.col2)

-- Remove table_label from an item.
openGauss=# ALTER RESOURCE LABEL table_label REMOVE COLUMN(table_for_label.col1);
```

Helpful Links

CREATE RESOURCE LABEL and **DROP RESOURCE LABEL**

11.14.23 ALTER ROLE

Function

ALTER ROLE modifies role attributes.

Precautions

None

Syntax

- Modify the permissions of a role.

  ```
  ALTER ROLE role_name [ [ WITH ] option [ ... ] ];
  ```

  The **option** clause for granting permissions is as follows:

  ```
  {CREATEDB | NOCREATEDB}
      | {CREATEROLE | NOCREATEROLE}
      | {INHERIT | NOINHERIT}
      | {AUDITADMIN | NOAUDITADMIN}
      | {SYSADMIN | NOSYSADMIN}
      | {MONADMIN | NOMONADMIN}
      | {OPRADMIN | NOOPRADMIN}
      | {POLADMIN | NOPOLADMIN}
      | {USEFT | NOUSEFT}
      | {LOGIN | NOLOGIN}
      | {REPLICATION | NOREPLICATION}
      | {INDEPENDENT | NOINDEPENDENT}
      | {VCADMIN | NOVCADMIN}
      | {PERSISTENCE | NOPERSISTENCE}
      | CONNECTION LIMIT connlimit
  ```

```
    | [ ENCRYPTED | UNENCRYPTED ] PASSWORD 'password' [EXPIRED]
    | [ ENCRYPTED | UNENCRYPTED ] IDENTIFIED BY 'password' [ REPLACE 'old_password' | EXPIRED ]
    | [ ENCRYPTED | UNENCRYPTED ] PASSWORD { 'password' | DISABLE | EXPIRED }
    | [ ENCRYPTED | UNENCRYPTED ] IDENTIFIED BY { 'password' [ REPLACE 'old_password' ] |
DISABLE }
    | VALID BEGIN 'timestamp'
    | VALID UNTIL 'timestamp'
    | RESOURCE POOL 'respool'
    | PERM SPACE 'spacelimit'
    | PGUSER
```

- Rename a role.

```
ALTER ROLE role_name
    RENAME TO new_name;
```

- Lock or unlock.

```
ALTER ROLE role_name
    ACCOUNT { LOCK | UNLOCK };
```

- Set parameters for a role.

```
ALTER ROLE role_name [ IN DATABASE database_name ]
    SET configuration_parameter {{ TO | = } { value | DEFAULT } | FROM CURRENT};
```

- Reset parameters for a role.

```
ALTER ROLE role_name
    [ IN DATABASE database_name ] RESET {configuration_parameter|ALL};
```

Parameter Description

- **role_name**

  Specifies a role name.

  Value range: an existing username

- **IN DATABASE database_name**

  Modifies the parameters of a role in a specified database.

- **SET configuration_parameter**

  Sets parameters for a role. Session parameters modified by **ALTER ROLE** apply to a specified role and take effect in the next session triggered by the role.

  Value range:

  For details about the values of **configuration_parameter** and **value**, see **SET**.

  DEFAULT: clears the value of **configuration_parameter**. **configuration_parameter** will inherit the default value of the new session generated for the role.

  FROM CURRENT: uses the value of **configuration_parameter** of the current session.

- **RESET configuration_parameter/ALL**

  Clears the value of **configuration_parameter**. The statement has the same effect as that of **SET configuration_parameter TO DEFAULT**.

  Value range: **ALL** indicates that the values of all parameters are cleared.

- **ACCOUNT LOCK | ACCOUNT UNLOCK**
  - **ACCOUNT LOCK**: locks an account to prevent it from logging in to the database.
  - **ACCOUNT UNLOCK**: unlocks an account and allows the account to log in to the database.

- **PGUSER**

  In the current version, the **PGUSER** attribute of a role cannot be modified.

- **PASSWORD/IDENTIFIED BY** 'password'

  Resets or changes the user password. Except the initial user, other administrators and common users need to enter the correct old password when changing their own passwords. Only the initial user, the system administrator (**sysadmin**), or users who have the permission to create users (**CREATEROLE**) can reset the password of a common user without entering the old password. The initial user can reset passwords of system administrators. A system administrator cannot reset passwords of other system administrators. The user password should be enclosed by single quotation marks.
- **EXPIRED**

  Invalidates the password. Only initial users, system administrators (**sysadmin**), and users who have the permission to create users (**CREATEROLE**) can invalidate user passwords. System administrators can invalidate their own passwords or the passwords of other system administrators. The password of the initial user cannot be invalidated.

  The user whose password is invalid can log in to the database but cannot perform the query operation. The query operation can be performed only after the password is changed or the administrator resets the password.

For details about other parameters, see **Parameter Description** in **CREATE ROLE**.

Examples

See **Examples** in **CREATE ROLE**.

Helpful Links

CREATE ROLE, **DROP ROLE**, and **SET**

11.14.24 ALTER ROW LEVEL SECURITY POLICY

Function

ALTER ROW LEVEL SECURITY POLICY modifies an existing row-level access control policy, including the policy name and the users and expressions affected by the policy.

Precautions

Only the table owner or a system administrator can perform this operation.

Syntax

```
ALTER [ ROW LEVEL SECURITY ] POLICY [ IF EXISTS ] policy_name ON table_name RENAME TO
new_policy_name;

ALTER [ ROW LEVEL SECURITY ] POLICY policy_name ON table_name
    [ TO { role_name | PUBLIC } [, ...] ]
    [ USING ( using_expression ) ];
```

Parameter Description

- policy_name

  Specifies the name of a row-level access control policy.
- table_name

  Specifies the name of a table to which a row-level access control policy is applied.
- new_policy_name

  Specifies the new name of a row-level access control policy.
- role_name

  Specifies names of users affected by a row-level access control policy. PUBLIC indicates that the row-level access control policy will affect all users.
- using_expression

  Specifies an expression defined for a row-level access control policy. The return value is of the boolean type.

Examples

```
-- Create the data table all_data.
openGauss=# CREATE TABLE all_data(id int, role varchar(100), data varchar(100));

--Create a row-level access control policy to specify that the current user can view only their own data.
openGauss=# CREATE ROW LEVEL SECURITY POLICY all_data_rls ON all_data USING(role =
CURRENT_USER);
openGauss=# \d+ all_data
                               Table "public.all_data"
 Column |          Type          | Modifiers | Storage  | Stats target | Description
--------+------------------------+-----------+----------+--------------+-------------
 id     | integer                |           | plain    |              |
 role   | character varying(100) |           | extended |              |
 data   | character varying(100) |           | extended |              |
Row Level Security Policies:
    POLICY "all_data_rls"
      USING (((role)::name = "current_user"()))
Has OIDs: no
Location Nodes: ALL DATANODES
Options: orientation=row, compression=no

-- Change the name of the all_data_rls policy.
openGauss=# ALTER ROW LEVEL SECURITY POLICY all_data_rls ON all_data RENAME TO all_data_new_rls;

-- Change the users affected by the row-level access control policy.
openGauss=# ALTER ROW LEVEL SECURITY POLICY all_data_new_rls ON all_data TO alice, bob;
openGauss=# \d+ all_data
                               Table "public.all_data"
 Column |          Type          | Modifiers | Storage  | Stats target | Description
--------+------------------------+-----------+----------+--------------+-------------
 id     | integer                |           | plain    |              |
 role   | character varying(100) |           | extended |              |
 data   | character varying(100) |           | extended |              |
Row Level Security Policies:
    POLICY "all_data_new_rls"
      TO alice,bob
      USING (((role)::name = "current_user"()))
Has OIDs: no
Location Nodes: ALL DATANODES
Options: orientation=row, compression=no, enable_rowsecurity=true

-- Modify the expression defined for the access control policy.
openGauss=# ALTER ROW LEVEL SECURITY POLICY all_data_new_rls ON all_data USING (id > 100 AND role
= current_user);
openGauss=# \d+ all_data
```

```
                              Table "public.all_data"
 Column |          Type          | Modifiers | Storage  | Stats target | Description
--------+------------------------+-----------+----------+--------------+-------------
 id     | integer                |           | plain    |              |
 role   | character varying(100) |           | extended |              |
 data   | character varying(100) |           | extended |              |
Row Level Security Policies:
    POLICY "all_data_new_rls"
      TO alice,bob
      USING (((id > 100) AND ((role)::name = "current_user"())))
Has OIDs: no
Location Nodes: ALL DATANODES
Options: orientation=row, compression=no, enable_rowsecurity=true
```

Helpful Links

CREATE ROW LEVEL SECURITY POLICY and **DROP ROW LEVEL SECURITY POLICY**

11.14.25 ALTER SCHEMA

Function

ALTER SCHEMA alters the attributes of a schema.

Precautions

- Only the owner of a schema or users granted with the ALTER permission on the schema can run the **ALTER SCHEMA** command. The system administrator has this permission by default. To change the owner of a schema, you must be the owner of the schema or system administrator and a member of the new owner role.
- Only the initial user is allowed to change the owner of the **pg_catalog** system schema. Changing the names of the built-in system schemas may make some functions unavailable or even affect the normal running of the database. By default, the names of the built-in system schemas cannot be changed. To ensure forward compatibility, you can change the names of the built-in system schemas only when the system is being started or upgraded or when **allow_system_table_mods** is set to **on**.

Syntax

- Alter the tamper-proof attribute of a schema.
  ```
  ALTER SCHEMA schema_name { WITH | WITHOUT } BLOCKCHAIN
  ```
- Rename a schema.
  ```
  ALTER SCHEMA schema_name
      RENAME TO new_name;
  ```
- Change the owner of a schema.
  ```
  ALTER SCHEMA schema_name
      OWNER TO new_owner;
  ```

Parameter Description

- **schema_name**

  Specifies the name of an existing schema.

  Value range: an existing schema name.

- **RENAME TO new_name**

  Rename a schema. If a non-administrator user wants to change the schema name, the user must have the CREATE permission on the database.

  new_name: new name of the schema.

  Value range: a string. It must comply with the identifier naming convention.

- **OWNER TO new_owner**

  Change the owner of a schema. To do this as a non-administrator, you must be a direct or indirect member of the new owner role, and that role must have the CREATE permission on the database.

  new_owner: new owner of the schema.

  Value range: an existing username or role name.

- **{ WITH | WITHOUT } BLOCKCHAIN**

  Alters the tamper-proof attribute of a schema. Common row-store tables with the tamper-proof attribute are tamper-proof history tables, excluding foreign tables, temporary tables, and system catalogs. The tamper-proof attribute can be altered only when no table is contained in the schema. The tamper-proof attribute of the temporary table schema, the toast table schema, dbe_perf schema, and blockchain schema cannot be modified.

Examples

```
-- Create the ds schema.
openGauss=# CREATE SCHEMA ds;

-- Rename the current schema ds to ds_new.
openGauss=# ALTER SCHEMA ds RENAME TO ds_new;

-- Create user jack.
openGauss=# CREATE USER jack PASSWORD 'xxxxxxxxx';

-- Change the owner of ds_new to jack.
openGauss=# ALTER SCHEMA ds_new OWNER TO jack;

-- Delete user jack and schema ds_new.
openGauss=# DROP SCHEMA ds_new;
openGauss=# DROP USER jack;
```

Helpful Links

CREATE SCHEMA and DROP SCHEMA

11.14.26 ALTER SEQUENCE

Function

ALTER SEQUENCE modifies the parameters of an existing sequence.

Precautions

- Only the sequence owner or a user granted with the ALTER permission can run the **ALTER SEQUENCE** command. The system administrator has this permission by default. To modify a sequence owner, you must be the sequence owner or system administrator and a member of the new owner role.

- In the current version, you can modify only the owner, owning column, and maximum value. To modify other parameters, delete the sequence and create it again. Then, use the **Setval** function to restore parameter values.
- **ALTER SEQUENCE MAXVALUE** cannot be used in transactions, functions, and stored procedures.
- After the maximum value of a sequence is changed, the cache of the sequence in all sessions is cleared.
- If the LARGE identifier is used when a sequence is created, the LARGE identifier must be used when the sequence is altered.
- The **ALTER SEQUENCE** statement blocks the invocation of **nextval**, **setval**, **currval**, and **lastval**.

Syntax

- Change the owning column of a sequence.
```
ALTER [ LARGE ] SEQUENCE [ IF EXISTS ] name
    [MAXVALUE maxvalue | NO MAXVALUE | NOMAXVALUE | CACHE cache]
    [ OWNED BY { table_name.column_name | NONE } ] ;
```
- Change the owner of a sequence.
```
ALTER [ LARGE ] SEQUENCE [ IF EXISTS ] name OWNER TO new_owner;
```

Parameter Description

- name

  Specifies the name of the sequence to be modified.
- IF EXISTS

  Sends a notice instead of an error when you are modifying a nonexisting sequence.
- CACHE

  Specifies the number of sequences stored in the memory for quick access purposes. If this parameter is not specified, the old cache value is retained.
- OWNED BY

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

  If the sequence has been associated with another table before you use this option, the new association will overwrite the old one.

  The associated table and sequence must be owned by the same user and in the same schema.

  If **OWNED BY NONE** is used, all existing associations will be deleted.
- new_owner

  Specifies the username of the new owner of the sequence. To change the owner, you must also be a direct or indirect member of the new role, and this role must have **CREATE** permission on the sequence's schema.

Examples

```
-- Create an ascending sequence named serial, starting from 101.
openGauss=# CREATE SEQUENCE serial START 101;
```

```
-- Create a table and define default values.
openGauss=# CREATE TABLE T1(C1 bigint default nextval('serial'));

-- Change the owning column of serial to T1.C1.
openGauss=# ALTER SEQUENCE serial OWNED BY T1.C1;

-- Delete a sequence and a table.
openGauss=# DROP SEQUENCE serial cascade;
openGauss=# DROP TABLE T1;
```

Helpful Links

CREATE SEQUENCE and **DROP SEQUENCE**

11.14.27 ALTER SERVER

Function

ALTER SERVER adds, modifies, or deletes the parameters of an existing server. You can query existing servers from the **pg_foreign_server** system catalog. The current feature is a lab feature. Contact Huawei technical support before using it.

Precautions

Only the server owner or a user granted with the **ALTER** permission can run the **ALTER SERVER** command. The system administrator has this permission by default. To modify a server owner, you must be the server owner or system administrator and a member of the new owner role.

When multi-layer quotation marks are used for sensitive columns (such as **password** and **secret_access_key**) in OPTIONS, the semantics is different from that in the scenario where quotation marks are not used. Therefore, sensitive columns are not identified for anonymization.

Syntax

- Change the parameters for a foreign server.
  ```
  ALTER SERVER server_name [ VERSION 'new_version' ]
      [ OPTIONS ( {[ ADD | SET | DROP ] option ['value']} [, ... ] ) ];
  ```
  In **OPTIONS**, **ADD**, **SET**, and **DROP** are operations to be performed. If these operations are not specified, **ADD** operations will be performed by default. **option** and **value** are the parameters of the corresponding operation.
- Change the name of a foreign server.
  ```
  ALTER SERVER server_name
     RENAME TO new_name;
  ```

Parameter Description

- **server_name**

  Specifies the name of the server to be modified.
- **new_version**

  Specifies the new version of the server.
- **OPTIONS**

  Change options of the server. **ADD**, **SET**, and **DROP** are operations to be performed. If the operation is not set explicitly, **ADD** is used. The option name must be unique, and the name and value are also validated with the foreign data wrapper library of the server.

In addition to the connection parameters supported by libpq, the following parameters are provided:

- **fdw_startup_cost**

  Estimates the startup time required for a foreign table scan, including the time to establish a connection, analyze the request at the remote server, and generate a plan. The default value is **100**.

- **fdw_typle_cost**

  Specifies the additional consumption when each tuple is scanned on a remote server. The value specifies the extra consumption of data transmission between servers. The default value is **0.01**.

- **new_name**

  Specifies the new name of the server.

Helpful Links

CREATE SERVER and **DROP SERVER**

11.14.28 ALTER SESSION

Function

ALTER SESSION defines or modifies the conditions or parameters that affect the current session. Modified session parameters are kept until the current session is disconnected.

Precautions

- If the **START TRANSACTION** statement is not executed before the **SET TRANSACTION** statement, the transaction is ended instantly and the statement does not take effect.
- You can use the **transaction_mode(s)** method declared in the **START TRANSACTION** statement to avoid using the **SET TRANSACTION** statement.

Syntax

- Set transaction parameters of a session.
```
ALTER SESSION SET [ SESSION CHARACTERISTICS AS ] TRANSACTION
  { ISOLATION LEVEL { READ COMMITTED } | { READ ONLY  | READ WRITE } } [, ...] ;
```
- Set other running parameters of a session.
```
ALTER SESSION SET
  {{config_parameter { { TO  | =  }  { value | DEFAULT }
    | FROM CURRENT }}
    | TIME ZONE time_zone
    | CURRENT_SCHEMA schema
    | NAMES encoding_name
    | ROLE role_name PASSWORD 'password'
    | SESSION AUTHORIZATION { role_name PASSWORD 'password' | DEFAULT }
    | XML OPTION { DOCUMENT | CONTENT }
  } ;
```

Parameter Description

For details about the descriptions of parameters related to **ALTER SESSION**, see **Parameter Description** of the SET syntax.

Examples

```
-- Create the ds schema.
openGauss=# CREATE SCHEMA ds;

-- Set the search path of the schema.
openGauss=# SET SEARCH_PATH TO ds, public;

-- Set the time/date type to the traditional Postgres format (date before month).
openGauss=# SET DATESTYLE TO postgres, dmy;

-- Set the character code of the current session to UTF8.
openGauss=# ALTER SESSION SET NAMES 'UTF8';

-- Set the time zone to Berkeley of California.
openGauss=# SET TIME ZONE 'PST8PDT';

-- Set the time zone to Italy.
openGauss=# SET TIME ZONE 'Europe/Rome';

-- Set the current schema.
openGauss=# ALTER SESSION SET CURRENT_SCHEMA TO tpcds;

-- Set XML OPTION to DOCUMENT.
openGauss=# ALTER SESSION SET XML OPTION DOCUMENT;

-- Create the role joe, and set the session role to joe.
openGauss=# CREATE ROLE joe WITH PASSWORD 'xxxxxxxxx';
openGauss=# ALTER SESSION SET SESSION AUTHORIZATION joe PASSWORD 'xxxxxxxxx';

-- Switch to the default user.
openGauss=> ALTER SESSION SET SESSION AUTHORIZATION default;

-- Delete the ds schema.
openGauss=# DROP SCHEMA ds;

-- Delete the role joe.
openGauss=# DROP ROLE joe;
```

Helpful Links

SET

11.14.29 ALTER SUBSCRIPTION

Function

ALTER SUBSCRIPTION alters the attributes of a subscription specified in **CREATE SUBSCRIPTION**.

Precautions

Only the owner of a subscription can execute **ALTER SUBSCRIPTION**. The new owner must be the system administrator.

Syntax

- Update the connection information of a subscription.
  ```
  ALTER SUBSCRIPTION name CONNECTION 'conninfo'
  ```
- Update the name of the publication on the publisher side.
  ```
  ALTER SUBSCRIPTION name SET PUBLICATION publication_name [, ...]
  ```
- Enable a subscription.
  ```
  ALTER SUBSCRIPTION name ENABLE
  ```
- Update the attributes defined in **CREATE SUBSCRIPTION**.
  ```
  ALTER SUBSCRIPTION name SET ( subscription_parameter [= value] [, ... ] )
  ```
- Update the owner of a subscription.
  ```
  ALTER SUBSCRIPTION name OWNER TO new_owner
  ```
- Change the name of a subscription.
  ```
  ALTER SUBSCRIPTION name RENAME TO new_name
  ```

Parameter Description

- **name**

  Specifies the name of the subscription whose attributes are to be altered.
- **CONNECTION 'conninfo'**

  Alters the connection attributes initially set by **CREATE SUBSCRIPTION**.
- **ENABLE (boolean)**

  Specifies whether a subscription should be actively replicated, or whether it should be just set but not yet started. The default value is **true**.
- **SET (subscription_parameter [= value] [, ...])**

  Modifies the parameters set by **CREATE SUBSCRIPTION**. The allowed parameters are **slot_name** and **synchronous_commit**.
  - If **enabled** is set to **false** during subscription creation, **slot_name** is forcibly set to **NONE**, that is, null. In this case, the replication slot does not exist even if the value of **slot_name** is specified.
  - Change the value of **enabled** from **false** to **true**. When the subscription is enabled, the publication side is connected to create a replication slot. If you do not specify the value of **slot_name**, the default value (subscription name) is used.
  - If **enabled** is set to **true**, the subscription is in the normal state. In this case, **slot_name** cannot be left empty, but the name of the replication slot can be changed to a valid name.
- **new_owner**

  Specifies the username of the new owner of a subscription.
- **new_name**

  Specifies the new name of a subscription.

Examples

For details, see **Examples**.

Helpful Links

CREATE SUBSCRIPTION and **DROP SUBSCRIPTION**

11.14.30 ALTER SYNONYM

Function

ALTER SYNONYM modifies the attributes of the **SYNONYM** object.

Precautions

- Currently, only the owner of the **SYNONYM** object can be changed.
- Only the system administrator has the permission to modify the owner of the **SYNONYM** object.
- The new owner must have the **CREATE** permission on the schema where the **SYNONYM** object resides.

Syntax

```
ALTER SYNONYM synonym_name
    OWNER TO new_owner;
```

Parameter Description

- **synonym**

  Specifies the name of the synonym to be modified, which can contain the schema name.

  Value range: a string. It must comply with the identifier naming convention.

- **new_owner**

  Specifies the new owner of the **SYNONYM** object.

  Value range: a string. It must be a valid username.

Examples

```
-- Create synonym t1.
openGauss=#  CREATE OR REPLACE SYNONYM t1 FOR ot.t1;

-- Create user u1.
openGauss=# CREATE USER u1 PASSWORD 'user@111';

-- Change the owner of synonym t1 to u1.
openGauss=# ALTER SYNONYM t1 OWNER TO u1;

-- Delete synonym t1.
openGauss=# DROP SYNONYM t1;

-- Delete user u1.
openGauss=# DROP USER u1;
```

Helpful Links

CREATE SYNONYM and **DROP SYNONYM**

11.14.31 ALTER SYSTEM KILL SESSION

Function

ALTER SYSTEM KILL SESSION ends a session.

Precautions

None

Syntax

```
ALTER SYSTEM KILL SESSION 'session_sid, serial' [ IMMEDIATE ];
```

Parameter Description

- **session_sid, serial**

  Specifies the SID and SERIAL of a session (To obtain the values, see the example.)

- **IMMEDIATE**

  Specifies that a session will be ended instantly after the statement is executed.

Examples

```
-- Query session information.
openGauss=#
SELECT sa.sessionid AS sid,0::integer AS serial#,ad.rolname AS username FROM pg_stat_get_activity(NULL)
AS sa
LEFT JOIN pg_authid ad ON(sa.usesysid = ad.oid)WHERE sa.application_name <> 'JobScheduler';
       sid       | serial# | username
-----------------+---------+----------
 140131075880720 |       0 | omm
 140131025549072 |       0 | omm
 140131073779472 |       0 | omm
 140131071678224 |       0 | omm
 140131125774096 |       0 |
 140131127875344 |       0 |
 140131113629456 |       0 |
 140131094742800 |       0 |
(8 rows)

-- End the session whose SID is 140131075880720.
openGauss=#  ALTER SYSTEM KILL SESSION '140131075880720,0' IMMEDIATE;
```

11.14.32 ALTER TABLE

Function

ALTER TABLE modifies tables, including modifying table definitions, renaming tables, renaming specified columns in tables, renaming table constraints, setting table schemas, enabling or disabling row-level security policies, and adding or updating multiple columns.

Precautions

- The owner of a table, users granted with the ALTER permission on the table, or users granted with the ALTER ANY TABLE permission can run the **ALTER TABLE** statement. The system administrator has the permission to run the command by default. To modify the owner or schema of a table, you must be the table owner or system administrator and a member of the new owner role.
- The tablespace of a partitioned table cannot be modified, but the tablespace of the partition can be modified.

- The storage parameter **ORIENTATION** cannot be modified.
- Currently, **SET SCHEMA** can only set schemas to user schemas. It cannot set a schema to a system internal schema.
- Column-store tables support only the **PARTIAL CLUSTER KEY**, **UNIQUE**, and **PRIMARY KEY** table-level constraints, but do not support foreign key table-level constraints.
- In a column-store table, you can perform **ADD COLUMN**, **ALTER TYPE**, **SET STATISTICS**, **DROP COLUMN** operations, and change table name and space. The types of new and modified columns should be the **Data Types** supported by column-store. The **USING** option of **ALTER TYPE** only supports constant expression and expression involved in the column.
- The column constraints supported by column-store tables include **NULL**, **NOT NULL**, **DEFAULT** constant values, **UNIQUE**, and **PRIMARY KEY**. Only the **DEFAULT** value can be modified (by using **SET DEFAULT** and **DROP DEFAULT**). Currently, **NULL** and **NOT NULL** constraints cannot be modified.
- Auto-increment columns cannot be added, or a column whose **DEFAULT** value contains the **nextval()** expression cannot be added.
- Row-access control cannot be enabled for foreign tables and temporary tables.
- When you delete a **PRIMARY KEY** constraint by constraint name, the **NOT NULL** constraint is not deleted. If necessary, manually delete the **NOT NULL** constraint.
- When JDBC is used, the **DEFAULT** value can be set through **PrepareStatement**.
- If you add a column using **ADD COLUMN**, all existing rows in the table are initialized to the column's default value (**NULL** if no DEFAULT clause is specified).

  If no **DEFAULT** value is specified for the new column, **NULL** is used, and no full table update is triggered.

  If the new column has a **DEFAULT** value, the column must meet all the following requirements. Otherwise, the entire table is updated, leading to additional overheads and affecting online services.

  1. The data type is BOOL, BYTEA, SMALLINT, BIGINT, SMALLINT, INTEGER, NUMERIC, FLOAT, DOUBLE PRECISION, CHAR, VARCHAR, TEXT, TIMESTAMPTZ, TIMESTAMP, DATE, TIME, TIMETZ, or INTERVAL.

  2. The length of the **DEFAULT** value of the added column cannot exceed 128 bytes.

  3. The **DEFAULT** value of the added column does not contain the volatile function.

  4. The **DEFAULT** value is required and cannot be **NULL**.

  If you are not sure whether condition 3 is met, check whether the **provolatile** attribute of the function in the PG_RPOC system catalog is **'v'**.

Syntax

- Modify the definition of a table.

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name ) }
    action [, ... ];
```

  There are several clauses of **action**:

```
column_clause
    | ADD table_constraint [ NOT VALID ]
    | ADD table_constraint_using_index
    | VALIDATE CONSTRAINT constraint_name
    | DROP CONSTRAINT [ IF EXISTS ]  constraint_name [ RESTRICT | CASCADE ]
    | CLUSTER ON index_name
    | SET WITHOUT CLUSTER
    | SET ( {storage_parameter = value} [, ... ] )
    | RESET ( storage_parameter [, ... ] )
    | OWNER TO new_owner
    | SET TABLESPACE new_tablespace
    | SET {COMPRESS|NOCOMPRESS}
    | TO { GROUP groupname | NODE ( nodename [, ... ] ) }
    | ADD NODE ( nodename [, ... ] )
    | DELETE NODE ( nodename [, ... ] )
    | DISABLE TRIGGER [ trigger_name | ALL | USER ]
    | ENABLE TRIGGER [ trigger_name | ALL | USER ]
    | ENABLE REPLICA TRIGGER trigger_name
    | ENABLE ALWAYS TRIGGER trigger_name
    | DISABLE/ENABLE [ REPLICA | ALWAYS ] RULE
    | DISABLE ROW LEVEL SECURITY
    | ENABLE ROW LEVEL SECURITY
    | FORCE ROW LEVEL SECURITY
    | NO FORCE ROW LEVEL SECURITY
    | ENCRYPTION KEY ROTATION
    | INHERIT parents
    | NO INHERIT parents
    | OF type_name
    | NOT OF
    | REPLICA IDENTITY { DEFAULT | USING INDEX index_name | FULL | NOTHING }
```

NOTE

- **ADD table_constraint [NOT VALID]**

  Adds a table constraint.
- **ADD table_constraint_using_index**

  Adds a primary key constraint or unique constraint to a table based on the existing unique index.
- **VALIDATE CONSTRAINT constraint_name**

  Validates a check-class constraint created with the **NOT VALID** option, and scans the entire table to ensure that all rows meet the constraint. Nothing happens if the constraint is already marked valid.
- **DROP CONSTRAINT [IF EXISTS] constraint_name [RESTRICT | CASCADE]**

  Deletes a table constraint.
- **CLUSTER ON index_name**

  Selects the default index for future CLUSTER operations. Actually, the table is not re-clustered.
- **SET WITHOUT CLUSTER**

  Deletes the most recently used **CLUSTER** index from the table. This affects future **CLUSTER** operations that do not specify an index.
- **SET ({storage_parameter = value} [, ...])**

  Changes one or more storage parameters for the table.
- **RESET (storage_parameter [, ...])**

  Resets one or more storage parameters to their defaults. As with **SET**, a table rewrite might be needed to update the table entirely.
- **OWNER TO new_owner**

  Changes the owner of a table, sequence, or view to the specified user.
- **SET TABLESPACE new_tablespace**

  Changes the table's tablespace to the specified tablespace and moves the data files associated with the table to the new tablespace. Indexes on the table, if any, are not moved; but they can be moved separately with additional **SET TABLESPACE** option in **ALTER INDEX**.
- **SET {COMPRESS|NOCOMPRESS}**

  Sets the compression feature of a table. The table compression feature affects only the storage mode of data inserted in a batch subsequently and does not affect storage of existing data. Setting the table compression feature will result in the fact that there are both compressed and uncompressed data in the table. Row-store tables do not support compression.
- **TO { GROUP groupname | NODE (nodename [, ...]) }**

  The syntax is only available in extended mode (when GUC parameter **support_extended_features** is **on**). Exercise caution when enabling the mode. It is mainly used for tools like internal dilatation tools. Common users should not use the mode.
- **ADD NODE (nodename [, ...])**

  It is only available for internal scale-out tools. Common users should not use the syntax.
- **DELETE NODE (nodename [, ...])**

  It is only available for internal scale-in tools. Common users should not use the syntax.
- **DISABLE TRIGGER [trigger_name | ALL | USER]**

  Disables a single trigger specified by **trigger_name**, disables all triggers, or disables only user triggers (excluding internally generated constraint triggers, for example, deferrable unique constraint triggers and exclusion constraints triggers).

Exercise caution when using this function because data integrity cannot be ensured as expected if the triggers are not executed.

- **| ENABLE TRIGGER [trigger_name | ALL | USER]**

  Enables a single trigger specified by **trigger_name**, enables all triggers, or enables only user triggers.

- **| ENABLE REPLICA TRIGGER trigger_name**

  Determines that the trigger firing mechanism is affected by the configuration variable **session_replication_role**. When the replication role is **origin** (default value) or **local**, a simple trigger is fired.

  When **ENABLE REPLICA** is configured for a trigger, it is fired only when the session is in replica mode.

- **| ENABLE ALWAYS TRIGGER trigger_name**

  Determines that all triggers are fired regardless of the current replication mode.

- **| DISABLE/ENABLE [REPLICA | ALWAYS] RULE**

  Enables or disables a rule for tables. Disabled rules are still visible in the system, but are not applied during query rewriting. The **ON SELECT** rule cannot be disabled because it is related to the view implementation. Rules configured as **ENABLE REPLICA** are enabled only when the session is in replica mode, while those configured as **ENABLE ALWAYS** can be enabled regardless of the replica mode. Rule triggering is also affected by configuration variables in **session_replication_role**, which is similar to the preceding trigger setting.

- **| DISABLE/ENABLE ROW LEVEL SECURITY**

  Enables or disables row-level access control for a table.

  If row-level access control is enabled for a data table but no row-level access control policy is defined, the row-level access to the data table is not affected. If row-level access control for a table is disabled, the row-level access to the table is not affected even if a row-level access control policy has been defined. For details, see **CREATE ROW LEVEL SECURITY POLICY**.

- **| NO FORCE/FORCE ROW LEVEL SECURITY**

  Forcibly enables or disables row-level access control for a table.

  By default, the table owner is not affected by the row-level access control feature. However, if row-level access control is forcibly enabled, the table owner (excluding system administrators) will be affected. System administrators are not affected by any row-level access control policies.

- **| ENCRYPTION KEY ROTATION**

  Rotation of the transparent data encryption key.

  The data encryption key rotation of a table can be performed only when the transparent encryption function is enabled for the database and **enable_tde** of the table is set to **on**. After the key rotation operation is performed, the system automatically applies for a new KMS key. After the key rotation, the data encrypted using the old key is decrypted using the old key, and the newly written data is encrypted using the new key. To ensure the security of encrypted data, you can periodically update the key based on the amount of new data in the encryption table. It is recommended that the key be updated every two to three years.

- **INHERIT parent_table**

  Adds the target data table to a specified parent data table as a new child data table. After that, the query for the parent data table will contain the data in the target data table. Before being added as a child data table, the target data table must contain all the columns in the parent data table. These columns must have matching data categories, and if they have NOT NULL constraints in the parent data table, they must also have NOT NULL constraints in the child data table. For all CHECK constraints in the parent data table, there must be corresponding constraints in the child data table, unless the parent data table is marked as non-inheritable.

- **NO INHERIT parent_table**

  Generates the target data table from the child data table of a specified parent data table. Queries for the parent data table will no longer contain records generated from the target data table.

- **OF type_name**

  Joins a table to a composite type, which is similar to table creation by using the **CREATE TABLE OF** option. The name and type of a table column must exactly match those defined in the composite type, but the OID system column can be different. The table cannot be inherited from any other table. These restrictions ensure that the **CREATE TABLE OF** option allows the same table definition.

- **NOT OF**

  Removes the association between a table and a type.

- **REPLICA IDENTITY { DEFAULT | USING INDEX index_name | FULL | NOTHING }**

  Specifies the record level of old tuples in UPDATE and DELETE statements on a table in logical replication scenarios.

  - **DEFAULT** records the old value of the primary key column. If there is no primary key, **DEFAULT** does not record the old value.
  - **USING INDEX** records the old values of columns covered by the named indexes. These values must be unique, non-local, and non-deferrable, and contain the values of columns marked **NOT NULL**.
  - **FULL** records the old values of all columns in the row.
  - **NOTHING** does not record information in old rows.

  In logical replication scenarios, when the UPDATE and DELETE statements of a table are parsed, the parsed old tuples consist of the information recorded in this method. For tables with primary keys, this option can be set to **DEFAULT** or **FULL**. For a table without a primary key, set this parameter to **FULL**. Otherwise, the old tuple will be parsed as empty during decoding. You are not advised to set this parameter to **NOTHING** in common scenarios because old tuples are always parsed as empty.

  NOTE

  Even if **DEFAULT** or **USING INDEX** is specified, the old values of the columns in the current Ustore table may contain the old values of all columns in the row. This configuration option takes effect only when the old values involve TOAST values. For the Ustore table, the **NOTHING** option is invalid, and the actual effect is the same as that of **FULL**.

– There are several clauses of **column_clause**:

```
ADD [ COLUMN ] column_name data_type [ compress_mode ] [ COLLATE collation ]
[ column_constraint [ ... ] ]
| MODIFY column_name data_type
| MODIFY column_name [ CONSTRAINT constraint_name ] NOT NULL [ ENABLE ]
| MODIFY column_name [ CONSTRAINT constraint_name ] NULL
| DROP [ COLUMN ] [ IF EXISTS ] column_name [ RESTRICT | CASCADE ]
| ALTER [ COLUMN ] column_name [ SET DATA ] TYPE data_type [ COLLATE collation ] [ USING
expression ]
| ALTER [ COLUMN ] column_name { SET DEFAULT expression | DROP DEFAULT }
| ALTER [ COLUMN ] column_name { SET | DROP } NOT NULL
| ALTER [ COLUMN ] column_name SET STATISTICS [PERCENT] integer
| ADD STATISTICS (( column_1_name, column_2_name [, ...] ))
| DELETE STATISTICS (( column_1_name, column_2_name [, ...] ))
| ALTER [ COLUMN ] column_name SET ( {attribute_option = value} [, ... ] )
| ALTER [ COLUMN ] column_name RESET ( attribute_option [, ... ] )
| ALTER [ COLUMN ] column_name SET STORAGE { PLAIN | EXTERNAL | EXTENDED | MAIN }
```

NOTE

- **ADD [COLUMN] column_name data_type [compress_mode] [COLLATE collation] [column_constraint [...]]**

  Adds a column to a table. If a column is added with **ADD COLUMN**, all existing rows in the table are initialized with the column's default value (**NULL** if no **DEFAULT** clause is specified).
- **ADD ({ column_name data_type [compress_mode] } [, ...])**

  Adds columns in the table.
- **MODIFY ({ column_name data_type | column_name [CONSTRAINT constraint_name] NOT NULL [ENABLE] | column_name [CONSTRAINT constraint_name] NULL } [, ...])**

  Modifies the data type of an existing column in the table.
- **DROP [COLUMN] [IF EXISTS] column_name [RESTRICT | CASCADE]**

  Drops a column from a table. Indexes and constraints related to the column are automatically dropped. If an object not belonging to the table depends on the column, **CASCADE** must be specified, such as a view.

  The **DROP COLUMN** statement does not physically remove the column, but simply makes it invisible to SQL operations. Subsequent **INSERT** and **UPDATE** operations in the table will store a **NULL** value for the column. Therefore, column deletion takes a short period of time but does not immediately release the tablespace on the disks, because the space occupied by the deleted column is not recycled. The space will be recycled when **VACUUM** is executed.
- **ALTER [COLUMN] column_name [SET DATA] TYPE data_type [COLLATE collation] [USING expression]**

  Modifies the type of a column in a table. Indexes and simple table constraints on the column will automatically use the new data type by reparsing the originally supplied expression.

  ALTER TYPE requires an entire table be rewritten. This is an advantage sometimes, because it frees up unnecessary space from a table. For example, to recycle the space occupied by a deleted column, the fastest method is to use the following statement.

  ```
  ALTER TABLE table ALTER COLUMN anycol TYPE anytype;
  ```

  In this statement, **anycol** indicates any column existing in the table and **anytype** indicates the type of the prototype of the column. **ALTER TYPE** does not change the table except that the table is forcibly rewritten. In this way, the data that is no longer used is deleted.
- **ALTER [COLUMN] column_name { SET DEFAULT expression | DROP DEFAULT }**

  Sets or removes the default value for a column. The default values only apply to subsequent **INSERT** operations; they do not cause rows already in the table to change. Defaults can also be created for views, in which case they are inserted into **INSERT** statements on the view before the view's **ON INSERT** rule is applied.
- **ALTER [COLUMN] column_name { SET | DROP } NOT NULL**

  Changes whether a column is marked to allow null values or to reject null values. You can only use **SET NOT NULL** when the column contains no null values.
- **ALTER [COLUMN] column_name SET STATISTICS [PERCENT] integer**

  Specifies the per-column statistics-gathering target for subsequent **ANALYZE** operations. The target can be set in the range from 0 to 10000. Set it to **–1** to revert to using the default system statistics target.
- **{ADD | DELETE} STATISTICS ((column_1_name, column_2_name [, ...]))**

Adds or deletes the declaration of collecting multi-column statistics to collect multi-column statistics as needed when **ANALYZE** is performed for a table or a database. The statistics about a maximum of 32 columns can be collected at a time. You are not allowed to add or delete such declaration for system catalogs or foreign tables.

- **ALTER [COLUMN] column_name SET ({attribute_option = value} [, ...])**

  ALTER [COLUMN] column_name RESET (attribute_option [, ...])

  Sets or resets per-attribute options.

  Currently, the only defined per-attribute options are **n_distinct** and **n_distinct_inherited**. **n_distinct** affects statistics of a table, while **n_distinct_inherited** affects the statistics of the table and its subtables. Currently, only **SET/RESET n_distinct** is supported, and **SET/RESET n_distinct_inherited** is forbidden.

- **ALTER [COLUMN] column_name SET STORAGE { PLAIN | EXTERNAL | EXTENDED | MAIN }**

  Sets the storage mode for a column. This clause specifies whether this column is held inline or in a secondary table, and whether the data should be compressed. It is set only for row-store tables and is invalid for column-store tables. If it is set for column-store tables, an error will be displayed when the statement is executed. **SET STORAGE** itself does not change anything in the table. It sets the strategy to be pursued during future table updates.

■ **column_constraint** is as follows:
```
[ CONSTRAINT constraint_name ]
    { NOT NULL |
      NULL |
      CHECK ( expression ) |
      DEFAULT default_expr  |
      GENERATED ALWAYS AS ( generation_expr ) STORED |
      UNIQUE index_parameters |
      PRIMARY KEY index_parameters |
      ENCRYPTED WITH ( COLUMN_ENCRYPTION_KEY = column_encryption_key,
ENCRYPTION_TYPE = encryption_type_value )
|
      REFERENCES reftable [ ( refcolumn ) ] [ MATCH FULL | MATCH PARTIAL | MATCH
SIMPLE ]
         [ ON DELETE action ] [ ON UPDATE action ] }    [ DEFERRABLE | NOT DEFERRABLE
| INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

■ **compress_mode** of a column is as follows:
```
[ DELTA | PREFIX | DICTIONARY | NUMSTR | NOCOMPRESS ]
```

– **table_constraint_using_index** used to add the primary key constraint or unique constraint based on the unique index is as follows:
```
[ CONSTRAINT constraint_name ]
    { UNIQUE | PRIMARY KEY } USING INDEX index_name
    [ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

– **table_constraint** is as follows:
```
[ CONSTRAINT constraint_name ]
    { CHECK ( expression ) |
      UNIQUE ( column_name [, ... ] ) index_parameters |
      PRIMARY KEY ( column_name [, ... ] ) index_parameters |
      PARTIAL CLUSTER KEY ( column_name [, ... ]  }
      FOREIGN KEY ( column_name [, ... ] ) REFERENCES reftable [ ( refcolumn [, ... ] ) ]
        [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ] [ ON DELETE action ] [ ON UPDATE
action ] }
    [ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

index_parameters is as follows:
```
[ WITH ( {storage_parameter = value} [, ... ] ) ]
    [ USING INDEX TABLESPACE tablespace_name ]
```

- Rename a table. The renaming does not affect stored data.

```
ALTER TABLE [ IF EXISTS ] table_name
    RENAME TO new_table_name;
```

- Rename the specified column in the table.

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name )}
    RENAME [ COLUMN ] column_name TO new_column_name;
```

- Rename the constraint of the table.

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name ) }
    RENAME CONSTRAINT constraint_name TO new_constraint_name;
```

- Set the schema of the table.

```
ALTER TABLE [ IF EXISTS ] table_name
    SET SCHEMA new_schema;
```

NOTE

- The schema setting moves the table into another schema. Associated indexes and constraints owned by table columns are migrated as well. Currently, the schema for sequences cannot be changed. If the table has sequences, delete the sequences, and create them again or delete the ownership between the table and sequences. In this way, the table schema can be changed.
- To change the schema of a table, you must also have the CREATE permission on the new schema. To add the table as a new child of a parent table, you must own the parent table as well. To alter the owner, you must also be a direct or indirect member of the new owning role, and that role must have the CREATE permission on the table's schema. These restrictions enforce that the user can only recreate and delete the table. However, a system administrator can alter the ownership of any table anyway.
- All the actions except for RENAME and SET SCHEMA can be combined into a list of multiple alterations to apply in parallel. For example, it is possible to add several columns or alter the type of several columns in a single statement. This is useful with large tables, since only one pass over the tables need be made.
- Adding a CHECK or NOT NULL constraint will scan the table to validate that existing rows meet the constraint.
- Adding a column with a non-NULL default or changing the type of an existing column will rewrite the entire table. Rewriting a large table may take much time and temporarily needs doubled disk space.

- Add columns.

```
ALTER TABLE [ IF EXISTS ] table_name
    ADD ( { column_name data_type [ compress_mode ] [ COLLATE collation ] [ column_constraint
[ ... ] ]} [, ...] );
```

- Update columns.

```
ALTER TABLE [ IF EXISTS ] table_name
    MODIFY ( { column_name data_type | column_name [ CONSTRAINT constraint_name ] NOT NULL
[ ENABLE ] | column_name [ CONSTRAINT constraint_name ] NULL } [, ...] );
```

Parameter Description

- **IF EXISTS**

  Sends a notice instead of an error if no tables have identical names. The notice prompts that the table you are querying does not exist.

- **table_name [*] | ONLY table_name | ONLY (table_name)**

  table_name is the name of the table that you need to modify.

  If **ONLY** is specified, only the table is modified. If **ONLY** is not specified, the table and all subtables are modified. You can add the asterisk (*) option following the table name to specify that all subtables are scanned, which is the default operation.

- **constraint_name**

Specifies the name of an existing constraint to drop.

- **index_name**

  Specifies the name of an index.

- **storage_parameter**

  Specifies the name of a storage parameter.

  TDE options:

  – enable_tde (bool type)

    Whether to enable TDE for a table. Before enabling this function, ensure that **enable_tde** in "GUC Parameter" has been enabled, KMS has been enabled, and the database instance master key ID in **tde_cmk_id** in "GUC Parameter" has been correctly configured.

    This parameter supports only row-store tables. Column-store tables and temporary tables are not supported. The Ustore storage engine is not supported. This parameter can be modified only when **enable_tde** is specified during table creation. Switching the encryption switch status does not change the encryption algorithm and key information.

    Value range: **on** and **off on** indicates that transparent data encryption is enabled. After the value is switched from **off** to **on**, new data is automatically encrypted when being written to the data page, and old data is automatically encrypted when the data page is updated. **off** indicates that transparent data encryption is disabled. After the value is switched from **on** to **off**, newly written data is not encrypted, old encrypted data can be automatically decrypted when being read, and data is not encrypted when being written back to the data page.

    Default value: **off**

  The following option is added for creating an index:

  – parallel_workers (int type)

    Number of bgworker threads started when an index is created. For example, value **2** indicates that two bgworker threads are started to create indexes concurrently.

    Value range: [0,32]. The value **0** indicates that this function is disabled.

    Default value: If this parameter is not set, the concurrent index creation function is disabled.

  – hasuids (Boolean type)

    Default value: **off**

    If this parameter is set to **on**, a unique table-level ID is allocated to a tuple when the tuple is updated.

  The following option is added to fix optimizer statistics:

  – min_tuples (float8)

    Default value: **0**

    The optimizer selects the larger value of the estimated statistics and the parameter to calculate the data volume based on the estimation table of statistics.

- **new_owner**

  Specifies the name of the new table owner.

- **new_tablespace**

  Specifies the new name of the tablespace to which the table belongs.
- column_name, column_1_name, column_2_name

  Specifies the name of a new or existing column.
- **data_type**

  Specifies the type of a new column or a new type of an existing column.
- **compress_mode**

  Compression option of a table field. The clause specifies the compression algorithm preferentially used by the column. Row-store tables do not support compression.
- **collation**

  Specifies the collation rule name of a column. The optional **COLLATE** clause specifies a collation for the new column; if omitted, the collation is the default for the new column. You can run the **select * from pg_collation;** command to query collation rules from the **pg_collation** system catalog. The default collation rule is the row starting with **default** in the query result.
- **USING expression**

  Specifies how to compute the new column value from the old; if omitted, the default conversion is an assignment cast from old data type to new. A **USING** clause must be provided if there is no implicit or assignment cast from the old to new type.

  NOTE

  USING in **ALTER TYPE** can specify any expression involving the old values of the row; that is, it can refer to any columns other than the one being cast. This allows general casting to be done with the **ALTER TYPE** syntax. Because of this flexibility, the **USING** expression is not applied to the column's default value (if any); the result might not be a constant expression as required for a default. This means that when there is no implicit or assignment cast from old to new type, **ALTER TYPE** might fail to convert the default even though a **USING** clause is supplied. In such cases, drop the default with **DROP DEFAULT**, perform the **ALTER TYPE**, and then use **SET DEFAULT** to add a suitable new default. Similar considerations apply to indexes and constraints involving the column.
- **NOT NULL | NULL**

  Sets whether the column allows null values.
- **integer**

  Specifies the constant value of a signed integer. When using **PERCENT**, the range of **integer** is from 0 to 100.
- **attribute_option**

  Specifies an attribute option.
- **PLAIN | EXTERNAL | EXTENDED | MAIN**

  Specifies a column-store mode.
  - **PLAIN** must be used for fixed-length values (such as integers). It must be inline and uncompressed.
  - **MAIN** is for inline, compressible data.
  - **EXTERNAL** is for external, uncompressed data. Use of **EXTERNAL** will make substring operations on **text** and **bytea** values run faster, at the penalty of increased storage space.

  - **EXTENDED** is for external, compressed data. **EXTENDED** is the default for most data types that support non-**PLAIN** storage.

- **CHECK (expression)**

  New rows or rows to be updated must satisfy for an expression to be true. If any row produces a false result, an error is raised and the database is not modified.

  A check constraint specified as a column constraint should reference only the column's values, while an expression in a table constraint can reference multiple columns.

  Currently, **CHECK (expression)** does not include subqueries and cannot use variables apart from the current column.

- **DEFAULT default_expr**

  Assigns a default data value to a column.

  The data type of the default expression must match the data type of the column.

  The default expression will be used in any insert operation that does not specify a value for the column. If there is no default value for a column, then the default value is null.

- **GENERATED ALWAYS AS (generation_expr) STORED**

  This clause creates a column as a generated column. The value of the generated column is calculated by **generation_expr** when data is written (inserted or updated). **STORED** indicates that the value of the generated column is stored as a common column.

  NOTE

  - The generation expression cannot refer to data other than the current row in any way. The generation expression cannot reference other generation columns or system columns. The generation expression cannot return a result set. No subquery, aggregate function, or window function can be used. The function called by the generation expression can only be an immutable function.
  - Default values cannot be specified for generated columns.
  - The generated column cannot be used as a part of the partition key.
  - Do not specify the generated column and the CASCADE, SET NULL, and SET DEFAULT actions of the ON UPDATE constraint at the same time. Do not specify the generated column and the SET NULL, and SET DEFAULT actions of the ON DELETE constraint at the same time.
  - The method of modifying and deleting generated columns is the same as that of common columns. Delete the common column that the generated column depends on. The generated column is automatically deleted. The type of the column on which the generated column depends cannot be changed.
  - The generated column cannot be directly written. In the INSERT or UPDATE statement, values cannot be specified for generated columns, but the keyword DEFAULT can be specified.
  - The permission control for generated columns is the same as that for common columns.
  - Columns cannot be generated for column-store tables and memory-optimized tables (MOTs). In foreign tables, only **postgres_fdw** supports generated columns.

- **UNIQUE index_parameters**

  UNIQUE (column_name [, ...]) index_parameters

Specifies that a group of one or more columns of a table can contain only unique values.

- **PRIMARY KEY index_parameters**

  PRIMARY KEY (column_name [, ...]) index_parameters

  Specifies that a column or columns of a table can contain only unique (non-duplicate) and non-null values.

- **REFERENCES reftable [(refcolum)] [MATCH matchtype] [ON DELETE action] [ON UPDATE action] (column constraint)**

  FOREIGN KEY (column_name [, ...]) REFERENCES reftable [(refcolumn [, ...])] [MATCH matchtype] [ON DELETE action] [ON UPDATE action] (table constraint)

  The foreign key constraint requires that the group consisting of one or more columns in the new table should contain and match only the referenced column values in the referenced table. If **refcolum** is omitted, the primary key of **reftable** is used. The referenced column should be the only column or primary key in the referenced table. A foreign key constraint cannot be defined between a temporary table and a permanent table.

  There are three types of matching between a reference column and a referenced column:

  – **MATCH FULL**: A column with multiple foreign keys cannot be **NULL** unless all foreign key columns are **NULL**.

  – **MATCH SIMPLE** (default): Any unexpected foreign key column can be **NULL**.

  – **MATCH PARTIAL**: This option is not supported currently.

  In addition, when you perform certain operations on the data in the referenced table, the operations are performed on the corresponding columns in the new table. **ON DELETE**: specifies the operations to be executed after a referenced row in the referenced table is deleted. **ON UPDATE**: specifies the operation to be performed when the referenced column data in the referenced table is updated. The possible actions of the **ON DELETE** and **ON UPDATE** clauses are as follows:

  – **NO ACTION** (default): When a foreign key is deleted or updated, an error indicating that the foreign key constraint is violated is created. If the constraint is deferrable and there are still any referenced rows, this error will occur when the constraint is checked.

  – **RESTRICT**: When a foreign key is deleted or updated, an error indicating that the foreign key constraint is violated is created. It is the same as **NO ACTION** except that the action cannot be delayed.

  – **CASCADE**: deletes any row that references the deleted row from the new table, or update the field value of the referenced row in the new table to the new value of the referenced column.

  – **SET NULL**: sets the referenced field to **NULL**.

  – **SET DEFAULT**: sets the referenced columns to their default values.

- **DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE**

  Sets whether the constraint can be deferrable.

  – **DEFERRABLE**: deferrable to the end of the transaction and checked using **SET CONSTRAINTS**.

  - **NOT DEFERRABLE**: checks immediately after the execution of each command.
  - **INITIALLY IMMEDIATE**: checks immediately after the execution of each statement.
  - **INITIALLY DEFERRED**: checks when the transaction ends.

    NOTE

    Ustore tables do not support the **DEFERRABLE** and **INITIALLY DEFERRED** constraints.

- **PARTIAL CLUSTER KEY**

  Specifies a partial cluster key for storage. When importing data to a column-store table, you can perform local data sorting by specified columns (single or multiple).

- **WITH ({storage_parameter = value} [, ...])**

  Specifies an optional storage parameter for a table or an index.

- **tablespace_name**

  Specifies the name of the tablespace where the index locates.

- **COMPRESS|NOCOMPRESS**
  - **NOCOMPRESS**: If the **NOCOMPRESS** keyword is specified, the existing compression feature of the table will not be changed.
  - **COMPRESS**: If the **COMPRESS** keyword is specified, the table compression feature will be triggered by batch tuple insertion. Row-store tables do not support compression.

- **new_table_name**

  Specifies the new table name.

- **new_column_name**

  Specifies the new name of a specific column in a table.

- **new_constraint_name**

  Specifies the new name of a table constraint.

- **new_schema**

  Specifies the new schema name.

- **CASCADE**

  Automatically drops objects that depend on the dropped column or constraint (for example, views referencing the column).

- **RESTRICT**

  Refuses to drop the column or constraint if there are any dependent objects. This is the default processing.

- **schema_name**

  Specifies the schema name of a table.

Examples

See **Examples** in **CREATE TABLE**.

Helpful Links

CREATE TABLE and **DROP TABLE**

11.14.33 ALTER TABLE PARTITION

Function

ALTER TABLE PARTITION modifies table partitions, including adding, deleting, splitting, merging, clearing, swapping, and renaming partitions, moving partition tablespaces, and modifying partition attributes.

Precautions

- The tablespace of the added partition cannot be **PG_GLOBAL**.
- The name of the added partition must be different from the names of existing partitions in the partitioned table.
- The key value of the added partition must be consistent with the type of partition keys in the partitioned table.
- If a range partition is added, the key value of the added partition must be greater than the upper limit of the last range partition in the partitioned table.
- If a list partition is added, the key value of the added partition cannot be the same as that of an existing partition.
- Hash partitions cannot be added.
- If the number of partitions in the target partitioned table has reached the maximum (**1048575**), partitions cannot be added.
- If a partitioned table has only one partition, the partition cannot be deleted.
- Use **PARTITION FOR()** to choose partitions. The number of specified values in the brackets should be the same as the column number in customized partitions, and they must be consistent.
- The **Value** partitioned table does not support the **Alter Partition** operation.
- Column-store tables and row-store tables cannot be partitioned.
- Partitions cannot be added to an interval partitioned table.
- Hash partitioned tables do not support splitting, combination, addition, and deletion of partitions.
- List partitioned tables do not support partition splitting or partition combination.
- Only the owner of a partitioned table or users granted with the ALTER permission on the partitioned table can run the **ALTER TABLE PARTITION** command. The system administrator has the permission to run the command by default.

Syntax

- Modify the syntax of the table partition.
  ```
  ALTER TABLE [ IF EXISTS ] { table_name  [*] | ONLY table_name | ONLY ( table_name  )}
      action [, ... ];
  ```
  action indicates the following clauses for maintaining partitions. For the partition continuity when multiple clauses are used for partition maintenance,

GaussDB runs **DROP PARTITION** and then **ADD PARTITION**, and finally runs the rest clauses in sequence.

```
move_clause  |
exchange_clause  |
row_clause  |
merge_clause  |
modify_clause  |
split_clause  |
add_clause  |
drop_clause  |
truncate_clause
```

– The **move_clause** syntax is used to move the partition to a new tablespace.

```
MOVE PARTITION { partion_name | FOR ( partition_value [, ...] ) } TABLESPACE tablespacename
```

– The **exchange_clause** syntax is used to move the data from a general table to a specified partition.

```
EXCHANGE PARTITION { ( partition_name ) | FOR ( partition_value [, ...] ) }
    WITH TABLE {[ ONLY ] ordinary_table_name | ordinary_table_name * | ONLY
( ordinary_table_name )}
    [ { WITH | WITHOUT } VALIDATION ] [ VERBOSE ] [ UPDATE GLOBAL INDEX ]
```

The ordinary table and partition whose data is to be exchanged must meet the following requirements:

- The number of columns of the ordinary table is the same as that of the partition, and their information should be consistent, including: column name, data type, constraint, collation information, storage parameter, and compression information.
- The compression information of the ordinary table and partition should be consistent.
- The number and information of indexes of the ordinary table and partition should be consistent.
- The number and information of constraints of the ordinary table and partition should be consistent.
- An ordinary table cannot be a temporary table. A partitioned table can only be a range partitioned table, list partitioned table, hash partitioned table, or interval partitioned table.
- Ordinary tables and partitioned tables do not support dynamic data masking and row-level access control constraints.
- List partitioned tables and hash partitioned tables cannot be column-store.

NOTICE

- When the exchange is done, the data and tablespace of the ordinary table and partition are exchanged. The statistics about ordinary tables and partitions become unreliable, and they should be analyzed again.
- A non-partition key cannot be used to create a local unique index. Therefore, if an ordinary table contains a unique index, data cannot be exchanged.

– The **row_clause** syntax is used to set row movement of a partitioned table.
```
{ ENABLE | DISABLE } ROW MOVEMENT
```
– The **merge_clause** syntax is used to merge partitions into one. Currently, only range partitions can be merged.
```
MERGE PARTITIONS { partition_name } [, ...] INTO PARTITION partition_name
    [ TABLESPACE tablespacename ] [ UPDATE GLOBAL INDEX ]
```

CAUTION

Ustore tables do not support ALTER TABLE MERGE PARTITIONS in transaction blocks.

– The **modify_clause** syntax is used to set whether a partition index is usable.
```
MODIFY PARTITION partition_name { UNUSABLE LOCAL INDEXES | REBUILD UNUSABLE LOCAL
INDEXES }
```
– The **split_clause** syntax is used to split one partition into partitions. Currently, only range partitions can be split.
```
SPLIT PARTITION { partition_name | FOR ( partition_value [, ...] ) } { split_point_clause |
no_split_point_clause } [ UPDATE GLOBAL INDEX ]
```
  ■ The **split_point_clause** syntax is used to specify a split point.
  ```
  AT ( partition_value ) INTO ( PARTITION partition_name [ TABLESPACE
  tablespacename ] , PARTITION partition_name [ TABLESPACE tablespacename ] )
  ```

  NOTICE

  - Column-store tables and row-store tables cannot be partitioned.
  - The size of the split point should be in the range of partition keys of the partition to be split. The split point can only split one partition into two new partitions.

  ■ The **no_split_point_clause** syntax does not specify a split point.
  ```
  INTO { ( partition_less_than_item [, ...] ) | ( partition_start_end_item [, ...] ) }
  ```

NOTICE

- The first new partition key specified by **partition_less_than_item** should be greater than that of the previously split partition (if any), and the last partition key specified by **partition_less_than_item** should equal that of the partition being split.
- The first new partition key specified by **partition_start_end_item** should equal that of the former partition (if any), and the last partition key specified by **partition_start_end_item** should equal that of the partition being split.
- **partition_less_than_item** supports a maximum of 4 partition keys, while **partition_start_end_item** supports only one partition key. For details about the supported data types, see **PARTITION BY RANGE(parti...**.
- **partition_less_than_item** and **partition_start_end_item** cannot be used in the same statement.

■ The syntax of **partition_less_than_item** is as follows:

```
PARTITION partition_name VALUES LESS THAN ( { partition_value | MAXVALUE }  [, ...] )
    [ TABLESPACE tablespacename ]
```

■ The syntax of **partition_start_end_item** is as follows. For details about the constraints, see **partition_start_end_item syntax**.

```
PARTITION partition_name {
        {START(partition_value) END (partition_value) EVERY (interval_value)} |
        {START(partition_value) END ({partition_value | MAXVALUE})} |
        {START(partition_value)} |
        {END({partition_value | MAXVALUE})}
} [TABLESPACE tablespace_name]
```

– The **add_clause** syntax is used to add one or more partitions to a specified partitioned table.

```
ADD PARTITION ( partition_col1_name = partition_col1_value [, partition_col2_name = 
partition_col2_value ] [, ...] )
    [ LOCATION 'location1' ]
    [ PARTITION (partition_colA_name = partition_colA_value [, partition_colB_name = 
partition_colB_value ] [, ...] ) ]
    [ LOCATION 'location2' ]
ADD {partition_less_than_item | partition_start_end_item| partition_list_item }
```

The syntax of **partition_list_item** is as follows:

```
PARTITION partition_name VALUES (list_values_clause)
    [ TABLESPACE tablespacename ]
```

NOTICE

- **partition_list_item** supports only one partition key. For details about the data types supported by **partition_list_item**, see **PARTITION BY LIST(partit...**.
- Interval and hash partitioned tables do not support partition addition.

– The **drop_clause** syntax is used to remove a partition from a specified partitioned table.

```
DROP PARTITION  { partition_name | FOR (  partition_value [, ...] )  } [ UPDATE GLOBAL INDEX ]
```

NOTICE

- Hash partitioned table does not support partition deletion.
- If a partitioned table has only one partition, the partition cannot be deleted.

– The **truncate_clause** syntax is used to remove a specified partition from a partitioned table.
```
TRUNCATE PARTITION  { partition_name | FOR (  partition_value [, ...] )  } [ UPDATE GLOBAL INDEX ]
```

- The syntax for modifying the name of a partition is as follows:
```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name  )}
    RENAME PARTITION { partion_name | FOR ( partition_value [, ...] ) } TO partition_new_name;
```

Parameter Description

- **table_name**

  Specifies the name of a partitioned table.

  Value range: an existing partitioned table name.
- **partition_name**

  Specifies the name of a partition.

  Value range: an existing partition name.
- **tablespacename**

  Specifies which tablespace the partition moves to.

  Value range: an existing tablespace name.
- **partition_value**

  Specifies the key value of a partition.

  The value specified by **PARTITION FOR (partition_value [, ...])** can uniquely identify a partition.

  Value range: partition keys for the partition to be renamed.
- **UNUSABLE LOCAL INDEXES**

  Sets all the indexes unusable in the partition.
- **REBUILD UNUSABLE LOCAL INDEXES**

  Rebuilds all the indexes in the partition.
- **ENABLE/DISABLE ROW MOVEMET**

  Sets row movement.

  If the tuple value is updated on the partition key during the **UPDATE** action, the partition where the tuple is located is altered. Setting this parameter enables error messages to be reported or movement of the tuple between partitions.

  Value range:

  – **ENABLE**: Row movement is enabled.

  – **DISABLE**: Row movement is disabled.

  The default value is **ENABLE**.
- **ordinary_table_name**

  Specifies the name of the ordinary table whose data is to be migrated.

Value range: an existing table name.

- **{ WITH | WITHOUT } VALIDATION**

  Checks whether the ordinary table data meets the specified partition key range of the partition to be migrated.

  Value range:

  – **WITH**: checks whether the ordinary table data meets the partition key range of the partition to be migrated. If any data does not meet the required range, an error is reported.

  – **WITHOUT**: does not check whether the ordinary table data meets the partition key range of the partition to be migrated.

  The default value is **WITH**.

  The check is time consuming, especially when the data volume is large. Therefore, use **WITHOUT** when you are sure that the current ordinary table data meets the partition key range of the partition to be migrated.

- **VERBOSE**

  When **VALIDATION** is **WITH**, if the ordinary table contains data that is out of the partition key range, insert the data to the correct partition. If there is no correct partition where the data can be inserted to, an error is reported.

  NOTICE

  Only when **VALIDATION** is **WITH**, **VERBOSE** can be specified.

- **partition_new_name**

  Specifies the new name of a partition.

  Value range: a string. It must comply with the identifier naming convention.

Examples

See **Examples** in **CREATE TABLE PARTITION**.

Helpful Links

CREATE TABLE PARTITION and **DROP TABLE**

11.14.34 ALTER TABLE SUBPARTITION

Function

ALTER TABLE SUBPARTITION modifies partitions from a level-2 partitioned table, including adding, deleting, clearing, and splitting partitions.

Precautions

- Currently, partitions from the level-2 partitioned table can be added, deleted, cleared, or split only.
- The tablespace of the added partition cannot be **PG_GLOBAL**.

- The name of the added partition must be different from the names of the existing level-1 and level-2 partitions in the partitioned table.
- The key value of the added partition must be consistent with the type of partition keys in the partitioned table.
- If a range partition is added, the key value of the added partition must be greater than the upper limit of the last range partition in the partitioned table. To add a partition to a table with the **MAXVALUE** partition, you are advised to use the **SPLIT** syntax.
- If a list partition is added, the key value of the added partition cannot be the same as that of an existing partition. To add a partition to a table with the **DEFAULT** partition, you are advised to use the **SPLIT** syntax.
- Hash partitions cannot be added. However, if the level-2 partition mode of an level-2 partitioned table is hash but the level-1 partition mode is not hash, you can add a level-1 partition and create the corresponding level-2 partition.
- If the number of partitions in the target partitioned table has reached the maximum (**1048575**), partitions cannot be added.
- If the partitioned table contains only one level-1 or level-2 partition, the partition cannot be deleted.
- Hash partitions cannot be deleted.
- Use **PARTITION FOR()** to choose partitions. The number of specified values in the brackets should be the same as the column number in customized partitions, and they must be consistent.
- Only level-2 partitions (leaf nodes) can be split. Only range and list partitioning policies can be used and hash partitioning policies are not supported. The list partitioning policy can be used only when the default partition is used.
- Only the owner of a partitioned table or users granted with the ALTER permission on the partitioned table can run the **ALTER TABLE PARTITION** command. The system administrator has the permission to run the command by default.
- If the ALTER statement does not contain UPDATE GLOBAL INDEX, the original GLOBAL index is invalid. In this case, other indexes are used for query. If the ALTER statement contains UPDATEGLOBAL INDEX, the original GLOBAL index is still valid and the index function is correct.

Syntax

- Modify the syntax of the table partition.
```
ALTER TABLE [ IF EXISTS ] { table_name  [*] | ONLY table_name | ONLY ( table_name  )}
    action [, ... ];
```
  action indicates the following clauses for maintaining partitions.
```
    add_clause    |
    drop_clause   |
    split_clause  |
    truncate_clause
```
- The **add_clause** syntax is used to add one or more partitions to a specified partitioned table. The syntax can be used in level-1 partitions.
```
ADD {partition_less_than_item | partition_list_item } [ ( subpartition_definition_list ) ]
```
  It can also be used in level-2 partitions.
```
MODIFY PARTITION partition_name ADD subpartition_definition
```
  partition_less_than_item defines a range partition. The syntax is as follows:

```
PARTITION partition_name VALUES LESS THAN ( partition_value | MAXVALUE ) [ TABLESPACE
tablespacename ]
```

partition_list_item defines a list partition. The syntax is as follows:

```
PARTITION partition_name VALUES ( partition_value [, ...] | DEFAULT ) [ TABLESPACE
tablespacename ]
```

subpartition_definition_list contains the **subpartition_definition** object of one or more level-2 partitions. The syntax is as follows:

```
SUBPARTITION subpartition_name [ VALUES LESS THAN ( partition_value | MAXVALUE ) | VALUES
( partition_value [, ...] | DEFAULT )]  [ TABLESPACE tablespace ]
```

NOTICE

If the level-1 partition is a hash partition, you cannot use **ADD** to add a level-1 partition. If the level-2 partition is a hash partition, you cannot use **MODIFY** to add a level-2 partition.

- The **drop_clause** syntax is used to remove a partition from a specified partitioned table. The syntax can be used in level-1 partitions.

```
DROP PARTITION  { partition_name | FOR (  partition_value )  } [ UPDATE GLOBAL INDEX ]
```

It can also be used in level-2 partitions.

```
DROP SUBPARTITION  { subpartition_name | FOR (  partition_value, subpartition_value )  } [ UPDATE
GLOBAL INDEX ]
```

NOTICE

- If the level-1 partition is a hash partition, the level-1 partition cannot be deleted. If the level-2 partition is a hash partition, the level-2 partition cannot be deleted.
- At least one sub-partition must be retained.

- The **split_clause** syntax is used to split one partition into different partitions.

```
SPLIT SUBPARTITION { subpartition_name} { split_point_clause  } [ UPDATE GLOBAL INDEX ]
```

The **split_point_clause** syntax used to specify a split point in the range partitioning policy is as follows:

```
AT ( subpartition_value ) INTO ( SUBPARTITION subpartition_name [ TABLESPACE tablespacename ] ,
SUBPARTITION subpartition_name [ TABLESPACE tablespacename ] )
```

The **split_point_clause** syntax used to specify a split point in the list partitioning policy is as follows:

```
VALUES ( subpartition_value ) INTO ( SUBPARTITION subpartition_name [ TABLESPACE
tablespacename ] , SUBPARTITION subpartition_name [ TABLESPACE tablespacename ] )
```

NOTICE

- The size of the split point should be in the range of the splitting partition key.
- One partition can be split into only two new partitions.
- In the range partitioning policy, the current partition is split into two partitions based on the split point. A partition smaller than the size specified by the split point is regarded as one partition, and a partition larger than the size specified by the split point is regarded as the other partition. Therefore, only one split point can be used in the range partitioning policy. In the list partitioning policy, there can be multiple but no more than 64 split points. These split points are extracted from the boundary values of the current partition as a new partition, and the remaining boundary values of the current partition are used as another new partition.
- Only the default list partition can be split.

- The **truncate_clause** syntax is used to remove a specified partition from a partitioned table.
```
TRUNCATE SUBPARTITION  { subpartition_name } [ UPDATE GLOBAL INDEX ]
```

Parameter Description

- **table_name**

  Specifies the name of a partitioned table.

  Value range: an existing partitioned table name.
- **subpartition_name**

  Specifies the name of a level-2 partition name.

  Value range: an existing level-2 partition name.
- **tablespacename**

  Specifies which tablespace the partition moves to.

  Value range: an existing tablespace name.

Examples

See the examples in **CREATE TABLE SUBPARTITION**.

11.14.35 ALTER TABLESPACE

Function

ALTER TABLESPACE modifies the attributes of a tablespace.

Precautions

- Only the tablespace owner or a user granted with the ALTER permission can run the **ALTER TABLESPACE** command. The system administrator has this permission by default. To modify a tablespace owner, you must be the tablespace owner or system administrator and a member of the new owner role.

- To change the owner, you must also be a direct or indirect member of the new owning role.

  NOTE

  If **new_owner** is the same as **old_owner**, the current user will not be verified. A message indicating successful **ALTER** execution is displayed.

Syntax

- The syntax of renaming a tablespace is as follows:
```
ALTER TABLESPACE tablespace_name
    RENAME TO new_tablespace_name;
```
- The syntax of setting the owner of a tablespace is as follows:
```
ALTER TABLESPACE tablespace_name
    OWNER TO new_owner;
```
- The syntax of setting the attributes of a tablespace is as follows:
```
ALTER TABLESPACE tablespace_name
    SET ( {tablespace_option = value} [, ... ] );
```
- The syntax of resetting the attributes of a tablespace is as follows:
```
ALTER TABLESPACE tablespace_name
    RESET ( { tablespace_option } [, ...] );
```
- The syntax of setting the quota of a tablespace is as follows:
```
ALTER TABLESPACE tablespace_name
    RESIZE MAXSIZE { UNLIMITED | 'space_size'};
```

Parameter Description

- **tablespace_name**

  Specifies the tablespace to be modified.

  Value range: an existing table name
- **new_tablespace_name**

  Specifies the new name of a tablespace.

  The new name cannot start with **PG_**.

  Value range: a string. It must comply with the naming convention.
- **new_owner**

  Specifies the new owner of the tablespace.

  Value range: an existing username
- **tablespace_option**

  Sets or resets the parameters of a tablespace.

  Value range:
  - seq_page_cost: sets the optimizer to calculate the cost of obtaining disk pages in sequence. The default value is **1.0**.
  - **random_page_cost**: sets the optimizer to calculate the cost of obtaining disk pages in a non-sequential manner. The default value is **4.0**.

NOTE

- The value of **random_page_cost** is relative to that of **seq_page_cost**. It is meaningless when the value is equal to or less than the value of **seq_page_cost**.
- The prerequisite for the default value **4.0** is that the optimizer uses indexes to scan table data and the hit ratio of table data in the cache is about 90%.
- If the size of the table data space is smaller than that of the physical memory, decrease the value to a proper level. On the contrary, if the hit ratio of table data in the cache is lower than 90%, increase the value.
- If random-access memory like SSD is adopted, the value can be decreased to a certain degree to reflect the cost of true random scan.

Value range: a positive floating point number

- **RESIZE MAXSIZE**

  Resets the maximum size of tablespace.

  Value range:

  - **UNLIMITED**: No limit is set for the tablespace.
  - Determined by **space_size**. For details about the format, see **CREATE TABLESPACE**.

  NOTE

  - If the adjusted quota is smaller than the current tablespace usage, the adjustment is successful. You need to decrease the tablespace usage to a value less than the new quota before writing data to the tablespace.
  - You can also use the following statement to change the value of **MAXSIZE**:

```
ALTER TABLESPACE tablespace_name RESIZE MAXSIZE
 { 'UNLIMITED' | 'space_size'};
```

Examples

See **Examples** in **CREATE TABLESPACE**.

Helpful Links

CREATE TABLESPACE and **DROP TABLESPACE**

11.14.36 ALTER TEXT SEARCH CONFIGURATION

Function

ALTER TEXT SEARCH CONFIGURATION modifies the definition of a text search configuration. You can modify its mappings from token types to dictionaries, change the configuration's name or owner, or modify the parameters.

The **ADD MAPPING FOR** form installs a list of dictionaries to be consulted for the specified token types; an error will be generated if there is already a mapping for any of the token types.

The **ALTER MAPPING FOR** form removes existing mapping for those token types and then adds specified mappings.

ALTER MAPPING REPLACE ... WITH ... and **ALTER MAPPING FOR... REPLACE ... WITH ...** options replace **old_dictionary** with **new_dictionary**. Note that only when **pg_ts_config_map** has tuples corresponding to **maptokentype** and **old_dictionary**, the update will succeed. If the update fails, no messages are returned.

The **DROP MAPPING FOR** form deletes all dictionaries for the specified token types in the text search configuration. If **IF EXISTS** is not specified and the string type mapping specified by **DROP MAPPING FOR** does not exist in text search configuration, an error will occur in the database.

Precautions

- If a search configuration is referenced (to create indexes), users are not allowed to modify the text search configuration.
- To use **ALTER TEXT SEARCH CONFIGURATION**, you must be the owner of the configuration.

Syntax

- Add text search configuration string mapping.

```
ALTER TEXT SEARCH CONFIGURATION name
    ADD MAPPING FOR token_type [, ... ] WITH dictionary_name [, ... ];
```

- Modify the text search configuration dictionary syntax.

```
ALTER TEXT SEARCH CONFIGURATION name
    ALTER MAPPING FOR token_type [, ... ] REPLACE old_dictionary WITH new_dictionary;
```

- Modify the text search configuration string.

```
ALTER TEXT SEARCH CONFIGURATION name
    ALTER MAPPING FOR token_type [, ... ] WITH dictionary_name [, ... ];
```

- Change the text search configuration dictionary.

```
ALTER TEXT SEARCH CONFIGURATION name
    ALTER MAPPING REPLACE old_dictionary WITH new_dictionary;
```

- Remove text search configuration string mapping.

```
ALTER TEXT SEARCH CONFIGURATION name
    DROP MAPPING [ IF EXISTS ] FOR token_type [, ... ];
```

- Rename the owner of text search configuration.

```
ALTER TEXT SEARCH CONFIGURATION name OWNER TO new_owner;
```

- Rename the text search configuration.

```
ALTER TEXT SEARCH CONFIGURATION name RENAME TO new_name;
```

- Rename the namespace of text search configuration.

```
ALTER TEXT SEARCH CONFIGURATION name SET SCHEMA new_schema;
```

- Modify the attributes of the text search configuration.

```
ALTER TEXT SEARCH CONFIGURATION name SET ( { configuration_option = value } [, ...] );
```

- Reset the attributes of text search configuration.

```
ALTER TEXT SEARCH CONFIGURATION name RESET ( {configuration_option} [, ...] );
```

Parameter Description

- **name**

  Specifies the name (optionally schema-qualified) of an existing text search configuration.
- **token_type**

Specifies the name of a token type that is emitted by the configuration's parser. For details, see **Parser**.

- **dictionary_name**

  Specifies the name of a text search dictionary. If multiple dictionaries are listed, they are searched in the specified order.

- **old_dictionary**

  Specifies the name of a text search dictionary to be replaced in the mapping.

- **new_dictionary**

  Specifies the name of a text search dictionary to be substituted for **old_dictionary**.

- **new_owner**

  Specifies the new owner of the text search configuration.

- **new_name**

  Specifies the new name of the text search configuration.

- **new_schema**

  Specifies the new schema for the text search configuration.

- **configuration_option**

  Specifies the text search configuration option. For details, see **CREATE TEXT SEARCH CONFIGURATION**.

- **value**

  Specifies the value of text search configuration option.

Examples

```
-- Create a text search configuration.
openGauss=# CREATE TEXT SEARCH CONFIGURATION english_1 (parser=default);
CREATE TEXT SEARCH CONFIGURATION

-- Add text search configuration string mapping.
openGauss=# ALTER TEXT SEARCH CONFIGURATION english_1 ADD MAPPING FOR word WITH
simple,english_stem;
ALTER TEXT SEARCH CONFIGURATION

-- Add text search configuration string mapping.
openGauss=# ALTER TEXT SEARCH CONFIGURATION english_1 ADD MAPPING FOR email WITH
english_stem, french_stem;
ALTER TEXT SEARCH CONFIGURATION

-- Query information about the text search configuration.
openGauss=# SELECT b.cfgname,a.maptokentype,a.mapseqno,a.mapdict,c.dictname FROM
pg_ts_config_map a,pg_ts_config b, pg_ts_dict c WHERE a.mapcfg=b.oid AND a.mapdict=c.oid AND
b.cfgname='english_1' ORDER BY 1,2,3,4,5;
  cfgname  | maptokentype | mapseqno | mapdict |   dictname
-----------+--------------+----------+---------+--------------
 english_1 |            2 |        1 |    3765 | simple
 english_1 |            2 |        2 |   12960 | english_stem
 english_1 |            4 |        1 |   12960 | english_stem
 english_1 |            4 |        2 |   12964 | french_stem
(4 rows)

-- Add text search configuration string mapping.
openGauss=# ALTER TEXT SEARCH CONFIGURATION english_1 ALTER MAPPING REPLACE french_stem with
german_stem;
ALTER TEXT SEARCH CONFIGURATION

-- Query information about the text search configuration.
```

```
openGauss=# SELECT b.cfgname,a.maptokentype,a.mapseqno,a.mapdict,c.dictname FROM
pg_ts_config_map a,pg_ts_config b, pg_ts_dict c WHERE a.mapcfg=b.oid AND a.mapdict=c.oid AND
b.cfgname='english_1' ORDER BY 1,2,3,4,5;
  cfgname  | maptokentype | mapseqno | mapdict |   dictname
-----------+--------------+----------+---------+--------------
 english_1 |            2 |        1 |    3765 | simple
 english_1 |            2 |        2 |   12960 | english_stem
 english_1 |            4 |        1 |   12960 | english_stem
 english_1 |            4 |        2 |   12966 | german_stem
(4 rows)
```

See **Examples** in **CREATE TEXT SEARCH CONFIGURATION**.

Helpful Links

CREATE TEXT SEARCH CONFIGURATION and **DROP TEXT SEARCH CONFIGURATION**

11.14.37 ALTER TEXT SEARCH DICTIONARY

Function

ALTER TEXT SEARCH DICTIONARY modifies the definition of a full-text search dictionary, including its parameters, name, owner, and schema.

Precautions

- Predefined dictionaries do not support the **ALTER** operations.
- Only the owner of a dictionary or a system administrator can perform the **ALTER** operations.
- After a dictionary is created or modified, any modification to the customized dictionary definition file in the **filepath** directory does not affect the dictionary in the database. To use these modifications in the database, run the **ALTER TEXT SEARCH DICTIONARY** statement to update the definition file of the corresponding dictionary.

Syntax

- Modify the dictionary definition.
  ```
  ALTER TEXT SEARCH DICTIONARY name (
      option [ = value ] [, ... ]
  );
  ```
- Rename a dictionary.
  ```
  ALTER TEXT SEARCH DICTIONARY name RENAME TO new_name;
  ```
- Set the schema of the dictionary.
  ```
  ALTER TEXT SEARCH DICTIONARY name SET SCHEMA new_schema;
  ```
- Change the owner of the dictionary.
  ```
  ALTER TEXT SEARCH DICTIONARY name OWNER TO new_owner;
  ```

Parameter Description

- **name**

  Specifies the name of an existing dictionary. (If you do not specify a schema name, the dictionary in the current schema will be used.)

  Value range: an existing dictionary name

- **option**

  Specifies the parameter name to be modified. Each type of dictionaries has a template containing their custom parameters. Parameters function in a way irrelevant to their setting sequence. For details about the parameters, see **option**.

  NOTE

  - The value of **TEMPLATE** in the dictionary cannot be changed.
  - To specify a dictionary, specify both the dictionary definition file path (**FILEPATH**) and the file name.
  - The name of a dictionary definition file can contain only lowercase letters, digits, and underscores (_).

- **value**

  Specifies the new value of a parameter. If the equal sign (=) and *value* are omitted, the previous settings of the option are deleted and the default value is used.

  Value range: valid values defined by **option**.

- **new_name**

  Specifies the new name of a dictionary.

  Value range: a string, which complies with the identifier naming convention. A value can contain a maximum of 63 characters.

- **new_owner**

  Specifies the new owner of a dictionary.

  Value range: an existing username

- **new_schema**

  Specifies the new schema of a dictionary.

  Value range: an existing schema

Examples

```
-- Modify the definition of stop words in Snowball dictionaries. Retain the values of other parameters.
openGauss=# ALTER TEXT SEARCH DICTIONARY my_dict ( StopWords = newrussian, FilePath = 'file:///home/
dicts' );

-- Modify the Language parameter in Snowball dictionaries and delete the definition of stop words.
openGauss=# ALTER TEXT SEARCH DICTIONARY my_dict (Language = dutch, StopWords);

-- Update the dictionary definition and do not change any other content.
openGauss=# ALTER TEXT SEARCH DICTIONARY my_dict ( dummy );
```

Helpful Links

CREATE TEXT SEARCH DICTIONARY and **DROP TEXT SEARCH DICTIONARY**

11.14.38 ALTER TRIGGER

Function

ALTER TRIGGER renames a trigger.

NOTE

Currently, only the name can be modified.

Precautions

Only the owner of a table where the trigger is created and a system administrator can run the **ALTER TRIGGER** statement.

Syntax

```
ALTER TRIGGER trigger_name ON table_name RENAME TO new_name;
```

Parameter Description

- **trigger_name**

  Specifies the name of the trigger to be modified.

  Value range: an existing trigger

- **table_name**

  Specifies the name of the table where the trigger to be modified is located.

  Value range: an existing table having a trigger

- **new_name**

  Specifies the new name after modification.

  Value range: a string, which complies with the identifier naming convention. A value contains a maximum of 63 characters and cannot be the same as other triggers on the same table.

Examples

See examples in **CREATE TRIGGER**.

Helpful Links

CREATE TRIGGER, **DROP TRIGGER**, and **ALTER TABLE**

11.14.39 ALTER TYPE

Function

ALTER TYPE modifies the definition of a type.

Precautions

Only the type owner or a user granted with the ALTER permission can run the **ALTER TYPE** command. The system administrator has this permission by default. To modify the owner or schema of a type, you must be a type owner or system administrator and a member of the new owner role.

Syntax

- Modify a type.

```
ALTER TYPE name action [, ... ]
ALTER TYPE name OWNER TO { new_owner | CURRENT_USER | SESSION_USER }
ALTER TYPE name RENAME ATTRIBUTE attribute_name TO new_attribute_name [ CASCADE |
RESTRICT ]
ALTER TYPE name RENAME TO new_name
ALTER TYPE name SET SCHEMA new_schema
ALTER TYPE name ADD VALUE [ IF NOT EXISTS ] new_enum_value [ { BEFORE | AFTER }
neighbor_enum_value ]
ALTER TYPE name RENAME VALUE existing_enum_value TO new_enum_value

where action is one of:
    ADD ATTRIBUTE attribute_name data_type [ COLLATE collation ] [ CASCADE | RESTRICT ]
    DROP ATTRIBUTE [ IF EXISTS ] attribute_name [ CASCADE | RESTRICT ]
    ALTER ATTRIBUTE attribute_name [ SET DATA ] TYPE data_type [ COLLATE collation ] [ CASCADE |
RESTRICT ]
```

- Add a new attribute to a composite type.
```
ALTER TYPE name ADD ATTRIBUTE attribute_name data_type [ COLLATE collation ] [ CASCADE |
RESTRICT ]
```
- Delete an attribute from a composite type.
```
ALTER TYPE name DROP ATTRIBUTE [ IF EXISTS ] attribute_name [ CASCADE | RESTRICT ]
```
- Change the type of an attribute in a composite type.
```
ALTER TYPE name ALTER ATTRIBUTE attribute_name [ SET DATA ] TYPE data_type [ COLLATE
collation ] [ CASCADE | RESTRICT ]
```
- Change the owner of a type.
```
ALTER TYPE name OWNER TO { new_owner | CURRENT_USER | SESSION_USER }
```
- Change the name of a type or the name of an attribute in a composite type.
```
ALTER TYPE name RENAME TO new_name
ALTER TYPE name RENAME ATTRIBUTE attribute_name TO new_attribute_name [ CASCADE |
RESTRICT ]
```
- Move a type to a new schema.
```
ALTER TYPE name SET SCHEMA new_schema
```
- Add a new value to an enumerated type.
```
ALTER TYPE name ADD VALUE [ IF NOT EXISTS ] new_enum_value [ { BEFORE | AFTER }
neighbor_enum_value ]
```
- Change an enumerated value in the value list.
```
ALTER TYPE name RENAME VALUE existing_enum_value TO new_enum_value
```

Parameter Description

- **name**

  Specifies the name of an existing type that needs to be modified (optionally schema-qualified).
- **new_name**

  Specifies the new name of the type.
- **new_owner**

  Specifies the new owner of the type.
- **new_schema**

  Specifies the new schema of the type.
- **attribute_name**

  Specifies the name of the attribute to be added, modified, or deleted.
- **new_attribute_name**

  Specifies the new name of the attribute to be renamed.
- **data_type**

Specifies the data type of the attribute to be added, or the new type of the attribute to be modified.

- **new_enum_value**

  Specifies a new enumerated value. It is a non-null string with a maximum length of 63 bytes.

- **neighbor_enum_value**

  Specifies an existing enumerated value before or after which a new enumerated value will be added.

- **existing_enum_value**

  Specifies an enumerated value to be changed. It is a non-null string with a maximum length of 63 bytes.

- **CASCADE**

  Determines that the type to be modified, its associated records, and subtables that inherit the type will all be updated.

- **RESTRICT**

  Refuses to update the associated records of the modified type. This is the default action.

NOTICE

- **ADD ATTRIBUTE**, **DROP ATTRIBUTE**, and **ALTER ATTRIBUTE** can be combined for processing. For example, it is possible to add several attributes or change the types of several attributes at the same time in one command.
- To modify a schema of a type, you must have the **CREATE** permission on the new schema. To change the owner, you must be a direct or indirect member of the new owning role, and the member must have the **CREATE** permission on the schema of this type. (These restrictions enforce that the user can only recreate and delete the type. However, the system administrator can change ownership of any type in any way.) To add an attribute or modify the type of an attribute, you must also have the **USAGE** permission of this type.

Examples

See **Examples** in **CREATE TYPE**.

Helpful Links

CREATE TYPE and **DROP TYPE**

11.14.40 ALTER USER

Function

ALTER USER modifies the attributes of a database user.

Precautions

Session parameters modified by **ALTER USER** apply to a specified user and take effect in the next session.

Syntax

- Modify user permissions or other information.
  ```
  ALTER USER user_name [ [ WITH ] option [ ... ] ];
  ```
  The **option** clause is as follows:
  ```
  { CREATEDB | NOCREATEDB }
      | { CREATEROLE | NOCREATEROLE }
      | { INHERIT | NOINHERIT }
      | { AUDITADMIN | NOAUDITADMIN }
      | { SYSADMIN | NOSYSADMIN }
      | {MONADMIN | NOMONADMIN}
      | {OPRADMIN | NOOPRADMIN}
      | {POLADMIN | NOPOLADMIN}
      | { USEFT | NOUSEFT }
      | { LOGIN | NOLOGIN }
      | { REPLICATION | NOREPLICATION }
      | {INDEPENDENT | NOINDEPENDENT}
      | {VCADMIN | NOVCADMIN}
      | {PERSISTENCE | NOPERSISTENCE}
      | CONNECTION LIMIT connlimit
      | [ ENCRYPTED | UNENCRYPTED ] PASSWORD { 'password' [EXPIRED] | DISABLE | EXPIRED }
      | [ ENCRYPTED | UNENCRYPTED ] IDENTIFIED BY { 'password' [ REPLACE 'old_password' |
  EXPIRED ] | DISABLE }
      | VALID BEGIN 'timestamp'
      | VALID UNTIL 'timestamp'
      | RESOURCE POOL 'respool'
      | PERM SPACE 'spacelimit'
      | PGUSER
  ```
- Change the username.
  ```
  ALTER USER user_name
      RENAME TO new_name;
  ```
- Lock or unlock.
  ```
  ALTER USER user_name
      ACCOUNT { LOCK | UNLOCK };
  ```
- Change the value of a specified parameter associated with the user.
  ```
  ALTER USER user_name
      SET configuration_parameter { { TO | = } { value | DEFAULT } | FROM CURRENT };
  ```
- Reset the value of a specified parameter associated with the user.
  ```
  ALTER USER user_name
      RESET { configuration_parameter | ALL };
  ```

Parameter Description

- **user_name**

  Specifies the current username.

  Value range: an existing username
- **new_password**

  Specifies a new password.

  The new password must:
  - Differ from the old password.
  - Contain at least eight characters. This is the default length.
  - Differ from the username or the username spelled backward.

  - Contain at least three types of the following four types of characters: uppercase characters (A to Z), lowercase characters (a to z), digits (0 to 9), and special characters, including: ~!@#$%^&*()-_=+\|[{}];:,<.>/?
  - Be enclosed by single quotation marks.

  Value range: a string

- **old_password**

  Specifies the old password.

- **ACCOUNT LOCK | ACCOUNT UNLOCK**
  - **ACCOUNT LOCK**: locks an account to forbid login to databases.
  - **ACCOUNT UNLOCK**: unlocks an account to allow login to databases.
- **PGUSER**

  In the current version, the **PGUSER** attribute of a user cannot be modified.

For details about other parameters, see "Parameter Description" in **CREATE ROLE** and **ALTER ROLE**.

Examples

See **Examples** in **CREATE USER**.

Helpful Links

CREATE ROLE, **CREATE USER**, and **DROP USER**

11.14.41 ALTER USER MAPPING

Function

ALTER USER MAPPING changes the definition of a user mapping.

Precautions

If the **password** option is displayed, ensure that the **usermapping.key.cipher** and **usermapping.key.rand** files exist in the *$GAUSSHOME*/**bin** directory of each node in GaussDB. If the two files do not exist, use the **gs_guc** tool to generate them and use the **gs_ssh** tool to release them to the *$GAUSSHOME*/**bin** directory on each node.

When multi-layer quotation marks are used for sensitive columns (such as **password**) in OPTIONS, the semantics is different from that in the scenario where quotation marks are not used. Therefore, sensitive columns are not identified for anonymization.

Syntax

```
ALTER USER MAPPING FOR { user_name | USER | CURRENT_USER | PUBLIC }
    SERVER server_name
    OPTIONS ( [ ADD | SET | DROP ] option ['value'] [, ... ] )
```

In **OPTIONS**, **ADD**, **SET**, and **DROP** are operations to be performed. If these operations are not specified, **ADD** operations will be performed by default. **option** and **value** are the parameters and values of the corresponding operation.

Parameter Description

- **user_name**

  Specifies user name of the mapping.

  CURRENT_USER and **USER** match the name of the current user. **PUBLIC** is used to match all current and future user names in the system.

- **server_name**

  Specifies name of the server to which the user is mapped.

- **OPTIONS**

  Changes an option for the user mapping. The new option overwrites any previously specified option. **ADD**, **SET**, and **DROP** are operations to be performed. If the operation is not set explicitly, **ADD** is used. The option name must be unique and will be validated with the foreign data wrapper of the server.

Helpful Links

CREATE USER MAPPING and DROP USER MAPPING

11.14.42 ALTER VIEW

Function

ALTER VIEW modifies all auxiliary attributes of a view. (To modify the query definition of a view, use **CREATE OR REPLACE VIEW**.)

Precautions

Only the view owner or a user granted with the ALTER permission can run the **ALTER VIEW** command. The system administrator has this permission by default. The following is permission constraints depending on attributes to be modified:

- To modify the schema of a view, you must be the owner of the view or system administrator and have the CREATE permission on the new schema.
- To modify the owner of a view, you must be the owner of the view or system administrator and a member of the new owner role, with the CREATE permission on the schema of the view.
- Do not change the type of a column in a view.

Syntax

- Set the default value of a view column.
```
ALTER VIEW [ IF EXISTS ] view_name
    ALTER [ COLUMN ] column_name SET DEFAULT expression;
```
- Remove the default value of a view column.
```
ALTER VIEW [ IF EXISTS ] view_name
    ALTER [ COLUMN ] column_name DROP DEFAULT;
```
- Change the owner of a view.
```
ALTER VIEW [ IF EXISTS ] view_name
    OWNER TO new_owner;
```
- Rename a view.
```
ALTER VIEW [ IF EXISTS ] view_name
    RENAME TO new_name;
```

- Set the schema of a view.
```
ALTER VIEW [ IF EXISTS ] view_name
    SET SCHEMA new_schema;
```
- Set the options of a view.
```
ALTER VIEW [ IF EXISTS ] view_name
    SET ( { view_option_name [ = view_option_value ] } [, ... ] );
```
- Reset the options of a view.
```
ALTER VIEW [ IF EXISTS ] view_name
    RESET ( view_option_name [, ... ] );
```

Parameter Description

- **IF EXISTS**

  If this option is used, no error is generated when the view does not exist, and only a message is displayed.
- **view_name**

  Specifies the view name, which can be schema-qualified.

  Value range: a string. It must comply with the naming convention rule.
- **column_name**

  Specifies an optional list of names to be used for columns of the view. If not given, the column names are deduced from the query.

  Value range: a string. It must comply with the naming convention.
- **SET/DROP DEFAULT**

  Sets or deletes the default value of a column. This parameter does not take effect.
- **new_owner**

  Specifies the new owner of a view.
- **new_name**

  Specifies the new view name.
- **new_schema**

  Specifies the new schema of the view.
- **view_option_name [= view_option_value]**

  Specifies an optional parameter for a view.

  Currently, **view_option_name** supports only the **security_barrier** parameter. This parameter is used when the view attempts to provide row-level security.

  Value range: Boolean type, **TRUE**, and **FALSE**.

Examples

```
-- Create a view consisting of rows with c_customer_sk less than 150.
openGauss=# CREATE VIEW tpcds.customer_details_view_v1 AS
    SELECT * FROM tpcds.customer
    WHERE c_customer_sk < 150;

-- Rename a view.
openGauss=# ALTER VIEW tpcds.customer_details_view_v1 RENAME TO customer_details_view_v2;

-- Change the schema of a view.
openGauss=# ALTER VIEW tpcds.customer_details_view_v2 SET schema public;

-- Delete a view.
openGauss=# DROP VIEW public.customer_details_view_v2;
```

Helpful Links

CREATE VIEW and **DROP VIEW**

11.14.43 ANALYZE | ANALYSE

Function

ANALYZE collects statistics about ordinary tables in a database, and stores the results in the **PG_STATISTIC** system catalog. The execution plan generator uses these statistics to determine which one is the most effective execution plan.

If no parameter is specified, **ANALYZE** analyzes each table and partitioned table in the current database. You can also specify the **table_name**, **column**, and **partition_name** parameters to restrict the analysis to a specific table, column, or partitioned table.

ANALYZE | ANALYSE VERIFY is used to check whether data files of common tables (row-store and column-store tables) in a database are damaged.

Precautions

- Non-temporary tables cannot be analyzed in an anonymous block, transaction block, function, or stored procedure. Temporary tables in a stored procedure can be analyzed but their statistics updates cannot be rolled back.
- The **ANALYZE VERIFY** operation is used to detect abnormal scenarios. The **RELEASE** version is required. In the **ANALYZE VERIFY** scenario, remote read is not triggered. Therefore, the remote read parameter does not take effect. If the system detects that a page is damaged due to an error in a key system table, the system directly reports an error and does not continue the detection.
- With no table specified, **ANALYZE** processes all the tables that the current user has permission to analyze in the current database. With tables specified, **ANALYZE** processes only the specified tables.
- To perform ANALYZE operation to a table, you must be a table owner or a user granted the VACUUM permission on the table. By default, the system administrator has this permission. However, database owners are allowed to **ANALYZE** all tables in their databases, except shared catalogs. (The restriction for shared catalogs means that a true database-wide **ANALYZE** can only be executed by the system administrator). **ANALYZE** skips tables on which users do not have permissions.

Syntax

- Collect statistics information about a table.
```
{ ANALYZE | ANALYSE } [ VERBOSE ]
    [ table_name [ ( column_name [, ...] ) ] ];
```
- Collect partition statistics information about a partitioned table. This syntax is not supported currently.
```
{ ANALYZE | ANALYSE } [ VERBOSE ]
    table_name [ ( column_name [, ...] ) ] PARTITION ( patrition_name ) ;
```

NOTE

An ordinary partitioned table supports the syntax but not the function of collecting statistics about specified partitions.

- Collect statistics about multiple columns. (The current feature is a lab feature. Contact Huawei technical support before using it.)
```
{ANALYZE | ANALYSE} [ VERBOSE ]
    table_name (( column_1_name, column_2_name [, ...] ));
```

NOTE

- When collecting statistics about multiple columns, set the GUC parameter **default_statistics_target** to a negative value to sample data in percentage.
- The statistics about a maximum of 32 columns can be collected at a time.
- You are not allowed to collect statistics about multiple columns in system catalogs.

- Check the data files in the current database.
```
{ANALYZE | ANALYSE} VERIFY {FAST|COMPLETE};
```

NOTE

- In fast mode, DML operations need to be performed on the tables to be verified concurrently. As a result, an error is reported during the verification. In the current fast mode, data is directly read from the disk. When other threads modify files concurrently, the obtained data is incorrect. Therefore, you are advised to perform the verification offline.
- You can perform operations on the entire database. Because a large number of tables are involved, you are advised to save the result **gsql -d database -p port -f "verify.sql"> verify_warning.txt 2>&1** in redirection mode.
- NOTICE is used to check only tables that are visible to external systems. The detection of internal tables is included in the external tables on which NOTICE depends and is not displayed externally.
- This statement can be executed with error tolerance. The **Assert** of the debug version may cause the core to fail to execute commands. Therefore, you are advised to perform the operations in release mode.
- If a key system table is damaged during a full database operation, an error is reported and the operation stops.

- Check data files of tables and indexes.
```
{ANALYZE | ANALYSE} VERIFY {FAST|COMPLETE} table_name|index_name [CASCADE];
```

NOTE

- Operations on ordinary tables and index tables are supported, but **CASCADE** operations on indexes of index tables are not supported. The **CASCADE** mode is used to process all index tables of the primary table. When the index tables are checked separately, the **CASCADE** mode is not required.
- When the primary table is checked, the internal tables of the primary table, such as the toast table and cudesc table, are also checked.
- When the system displays a message indicating that the index table is damaged, you are advised to run the **reindex** command to recreate the index.

- Check the data files of the table partition.
```
{ANALYZE | ANALYSE} VERIFY {FAST|COMPLETE} table_name PARTITION {(patrition_name)}
[CASCADE];
```

NOTE

You can check a single partition of a table, but cannot perform the **CASCADE** operation on the indexes of an index table.

Parameter Description

- **VERBOSE**

  Enables the display of progress messages.

  NOTE

  If **VERBOSE** is specified, **ANALYZE** displays the progress information, indicating the table that is being processed. Statistics about tables are also displayed.

- **table_name**

  Specifies the name (possibly schema-qualified) of a specific table to analyze. If omitted, all regular tables (but not foreign tables) in the current database are analyzed.

  Currently, you can use **ANALYZE** to collect statistics only from row-store tables and column-store tables.

  Value range: an existing table name

- **column_name**, column_1_name, column_2_name

  Specifies the name of a specific column to analyze. All columns are analyzed by default.

  Value range: an existing column name

- **partition_name**

  Assumes the table is a partitioned table. You can specify **partition_name** following the keyword **PARTITION** to analyze the statistics of this table. Currently, **ANALYZE** can be performed on partitioned tables, but statistics of specified partitions cannot be analyzed.

  Value range: a partition name of a table

- **index_name**

  Specifies the name of the specific index table to be analyzed (possibly schema-qualified).

  Value range: an existing table name

- **FAST|COMPLETE**

  For a row-store table, the **FAST** mode verifies the CRC and page header of the row-store table. If the verification fails, an alarm is generated. In **COMPLETE** mode, the pointer and tuple of the row-store table are parsed and verified. For a column-store table, the **FAST** mode verifies the CRC and magic of the column-store table. If the verification fails, an alarm is generated. In **COMPLETE** mode, the CU of the column-store table is parsed and verified.

- **CASCADE**

  In **CASCADE** mode, all indexes of the current table are verified.

Examples

-- Create a table.

```
openGauss=# CREATE TABLE customer_info
(
WR_RETURNED_DATE_SK       INTEGER                      ,
WR_RETURNED_TIME_SK       INTEGER                      ,
WR_ITEM_SK                INTEGER              NOT NULL,
WR_REFUNDED_CUSTOMER_SK   INTEGER
)
;
```

-- Create a partitioned table.

```
openGauss=# CREATE TABLE customer_par
(
WR_RETURNED_DATE_SK       INTEGER                       ,
WR_RETURNED_TIME_SK       INTEGER                       ,
WR_ITEM_SK                INTEGER               NOT NULL,
WR_REFUNDED_CUSTOMER_SK   INTEGER
)
PARTITION BY RANGE(WR_RETURNED_DATE_SK)
(
PARTITION P1 VALUES LESS THAN(2452275),
PARTITION P2 VALUES LESS THAN(2452640),
PARTITION P3 VALUES LESS THAN(2453000),
PARTITION P4 VALUES LESS THAN(MAXVALUE)
)
ENABLE ROW MOVEMENT;
```

-- Run **ANALYZE** to update statistics.

```
openGauss=# ANALYZE customer_info;
openGauss=# ANALYZE customer_par;
```

-- Run the **ANALYZE VERBOSE** statement to update statistics and display table information.

```
openGauss=# ANALYZE VERBOSE customer_info;
INFO:  analyzing "cstore.pg_delta_3394584009"(cn_5002 pid=53078)
INFO:  analyzing "public.customer_info"(cn_5002 pid=53078)
INFO:  analyzing "public.customer_info" inheritance tree(cn_5002 pid=53078)
ANALYZE
```

NOTE

If any environment-related fault occurs, check the logs of the primary node of the database.

-- Delete the table.

```
openGauss=# DROP TABLE customer_info;
openGauss=# DROP TABLE customer_par;
```

11.14.44 BEGIN

Function

BEGIN may be used to initiate an anonymous block or a single transaction. This section describes the syntax of **BEGIN** used to initiate an anonymous block. For details about the **BEGIN** syntax that initiates transactions, see **START TRANSACTION**.

An anonymous block is a structure that can dynamically create and execute stored procedure code instead of permanently storing code as a database object in the database.

Precautions

None

Syntax

- Enable an anonymous block.
```
[DECLARE [declare_statements]]
BEGIN
execution_statements
END;
/
```
- Start a transaction.
```
BEGIN [ WORK | TRANSACTION ]
  [
    {
      ISOLATION LEVEL { READ COMMITTED | SERIALIZABLE | REPEATABLE READ }
      | { READ WRITE | READ ONLY }
     } [, ...]
  ];
```

Parameter Description

- **declare_statements**

  Declares a variable, including its name and type, for example, **sales_cnt int**.
- **execution_statements**

  Specifies the statement to be executed in an anonymous block.

  Value range: DML operations (such as select, insert, delete, and update) or registered functions in the system catalog.

Examples

None

Helpful Links

START TRANSACTION

11.14.45 CALL

Function

CALL calls defined functions and stored procedures.

Precautions

The owner of a function or stored procedure, users granted with the **EXECUTE** permission on the function or stored procedure, or users granted with the **EXECUTE ANY FUNCTION** permission can call the function or stored procedure. The system administrator has the permission to call the function or stored procedure by default.

Syntax

```
CALL [schema.|package.] {func_name| procedure_name} ( param_expr );
```

Parameter Description

- **schema**

Specifies the name of the schema where a function or stored procedure is located.

- package

  Specifies the name of the package where a function or stored procedure is located.

- **func_name**

  Specifies the name of the function or stored procedure to be called.

  Value range: an existing function name.

- **param_expr**

  Specifies a list of parameters. Use := or => to separate a parameter name and its value. This method allows parameters to be placed in any order. If only parameter values are in the list, the value order must be the same as that defined in the function or stored procedure.

  Value range: an existing function parameter name or stored procedure parameter name.

  NOTE

  The parameters include input parameters (whose name and type are separated by IN) and output parameters (whose name and type are separated by OUT). When you run the **CALL** statement to call a function or stored procedure, the parameter list must contain an output parameter for non-overloaded functions. You can set the output parameter to a variable or any constant. For details, see **Examples**. For an overloaded package function, the parameter list can have no output parameter, but the function may not be found. If an output parameter is contained, it must be a constant.

Examples

```
-- Create the func_add_sql function, calculate the sum of two integers, and return the result.
openGauss=# CREATE FUNCTION func_add_sql(num1 integer, num2 integer) RETURN integer
AS
BEGIN
RETURN num1 + num2;
END;
/

-- Transfer by parameter value.
openGauss=# CALL func_add_sql(1, 3);

-- Transfer by naming tag method.
openGauss=# CALL func_add_sql(num1 => 1,num2 => 3);
openGauss=# CALL func_add_sql(num2 := 2, num1 := 3);

-- Delete the function.
openGauss=# DROP FUNCTION func_add_sql;

-- Create a function with output parameters.
openGauss=# CREATE FUNCTION func_increment_sql(num1 IN integer, num2 IN integer, res OUT integer)
RETURN integer
AS
BEGIN
res := num1 + num2;
END;
/

-- Set output parameters to constants.
openGauss=# CALL func_increment_sql(1,2,1);

-- Delete the function.
openGauss=# DROP FUNCTION func_increment_sql;
```

11.14.46 CHECKPOINT

Function

A checkpoint is a point in the transaction log sequence at which all data files have been updated to reflect the information in the log. All data files will be flushed to a disk.

CHECKPOINT forces a transaction log checkpoint. By default, WALs periodically specify checkpoints in a transaction log. You may use **gs_guc** to specify run-time parameters **checkpoint_segments**, **checkpoint_timeout**, and **incremental_checkpoint_timeout** to adjust the atomized checkpoint intervals.

Precautions

- Only the system administrator and O&M administrator can invoke **CHECKPOINT**.
- **CHECKPOINT** forces an immediate checkpoint when the related command is issued, without waiting for a regular checkpoint scheduled by the system.

Syntax

```
CHECKPOINT;
```

Parameter Description

None

Examples

```
-- Set a checkpoint.
openGauss=# CHECKPOINT;
```

11.14.47 CLEAN CONNECTION

Function

CLEAN CONNECTION clears database connections. You may use this statement to delete a specific user's connections to a specified database.

Precautions

- GaussDB does not support specified nodes and supports only TO ALL.
- This function can be used to clear the normal connections that are being used only in fore mode.

Syntax

```
CLEAN CONNECTION
     TO { COORDINATOR ( nodename [, ... ] ) | NODE ( nodename [, ... ] )| ALL [ CHECK ] [ FORCE ] }
     [ FOR DATABASE dbname ]
     [ TO USER username ];
```

Parameter Description

- **CHECK**

  This parameter can be specified only when the node list is specified as **TO ALL**. Setting this parameter will check whether a database is accessed by other sessions before its connections are cleared. If any sessions are detected before **DROP DATABASE** is executed, an error will be reported and the database will not be deleted.
- **FORCE**

  This parameter can be specified only when the node list is specified as **TO ALL**. Setting this parameter will send **SIGTERM** signals to all the threads related to the specified **dbname** and **username** and forcibly shut them down.
- **COORDINATOR (nodename [, ...]) | NODE (nodename [, ...]) | ALL**

  Only **TO ALL** is supported. This parameter must be specified. All specified connections on the node will be deleted.
- **dbname**

  Deletes connections to a specified database. If this parameter is not specified, connections to all databases will be deleted.

  Value range: an existing database name
- **username**

  Deletes connections of a specific user. If this parameter is not specified, connections of all users will be deleted.

  Value range: an existing username

Examples

```
-- Create user jack.
CREATE USER jack PASSWORD 'Bigdata123@';

-- Clean the user jack's connections to the template1 database.
CLEAN CONNECTION TO ALL FOR DATABASE template1 TO USER jack;

-- Delete all connections of user jack.
CLEAN CONNECTION TO ALL TO USER jack;

-- Clean all the connections to the gaussdb database.
CLEAN CONNECTION TO ALL FORCE FOR DATABASE gaussdb;

-- Delete user jack.
DROP USER jack;
```

11.14.48 CLOSE

Function

CLOSE frees the resources associated with an open cursor.

Precautions

- After a cursor is closed, no subsequent operations are allowed on it.
- A cursor should be closed when it is no longer needed.
- Every non-holdable open cursor is implicitly closed when a transaction is terminated by **COMMIT** or **ROLLBACK**.

- A holdable cursor is implicitly closed if the transaction that created it aborts by **ROLLBACK**.
- If the cursor creation transaction is successfully committed, the holdable cursor remains open until an explicit **CLOSE** operation is executed, or the client disconnects.
- GaussDB does not have an explicit **OPEN** cursor statement. A cursor is considered open when it is declared. You can view all available cursors by querying the **pg_cursors** system view.

Syntax

```
CLOSE { cursor_name | ALL } ;
```

Parameter Description

- **cursor_name**

  Specifies the name of a cursor to be closed.
- **ALL**

  Closes all open cursors.

Examples

See **Examples** in **FETCH**.

Helpful Links

FETCH and **MOVE**

11.14.49 CLUSTER

Function

CLUSTER is used to cluster a table based on an index.

CLUSTER instructs GaussDB to cluster the table specified by **table_name** based on the index specified by **index_name**. The index must have been defined by **table_name**.

When a table is clustered, it is physically reordered based on the index information. Clustering is a one-time operation. When the table is subsequently updated, the changes are not clustered. That is, no attempt is made to store new or updated rows according to their index order.

When a table is clustered, GaussDB records which index the table was clustered by. The form **CLUSTER table_name** reclusters the table using the same index as before. You can also use the **CLUSTER** or **SET WITHOUT CLUSTER** form of **ALTER TABLE** to set the index to be used for future cluster operations, or to clear any previous settings.

CLUSTER without any parameter reclusters all the previously-clustered tables in the current database that the calling user owns, or all such tables if called by an administrator.

When a table is being clustered, an **ACCESS EXCLUSIVE** lock is acquired on it. This prevents any other database operations (both read and write) from being performed on the table until the **CLUSTER** is finished.

Precautions

- Only row-store B-tree indexes support **CLUSTER**.
- In the case where you are accessing single rows randomly within a table, the actual order of the data in the table is unimportant. However, if you tend to access some data more than others, and there is an index that groups them together, it is helpful by using **CLUSTER**. If you are requesting a range of indexed values from a table, or a single indexed value that has multiple rows that match, **CLUSTER** will help because once the index identifies the table page for the first row that matches, all other rows that match are probably already on the same table page, and so you save disk accesses and speed up the query.
- When an index scan is used, a temporary copy of the table is created that contains the table data in the index order. Temporary copies of each index on the table are created as well. Therefore, you need free space on disk at least equal to the sum of the table size and the total index size.
- Because **CLUSTER** remembers which indexes are clustered, one can cluster the tables manually the first time, then set up a time like **VACUUM** without any parameters, so that the desired tables are periodically reclustered.
- Because the optimizer records statistics about the ordering of tables, it is advisable to run **ANALYZE** on the newly clustered table. Otherwise, the optimizer might make poor choices of query plans.
- **CLUSTER** cannot be executed in transactions.
- If the **xc_maintenance_mode** parameter is not enabled, the CLUSTER operation will skip all system catalogs.

Syntax

- Cluster a table.
  ```
  CLUSTER [ VERBOSE ] table_name [ USING index_name ];
  ```
- Cluster a partition.
  ```
  CLUSTER [ VERBOSE ] table_name PARTITION ( partition_name ) [ USING index_name ];
  ```
- Recluster a table.
  ```
  CLUSTER [ VERBOSE ];
  ```

Parameter Description

- **VERBOSE**

  Enables the display of progress messages.
- **table_name**

  Specifies the table name.

  Value range: an existing table name
- **index_name**

  Specifies the index name.

  Value range: an existing index name

- **partition_name**

  Specifies the partition name.

  Value range: an existing partition name

Examples

```
-- Create a partitioned table.
openGauss=# CREATE TABLE tpcds.inventory_p1
(
    INV_DATE_SK               INTEGER               NOT NULL,
    INV_ITEM_SK               INTEGER               NOT NULL,
    INV_WAREHOUSE_SK          INTEGER               NOT NULL,
    INV_QUANTITY_ON_HAND      INTEGER
)
PARTITION BY RANGE(INV_DATE_SK)
(
        PARTITION P1 VALUES LESS THAN(2451179),
        PARTITION P2 VALUES LESS THAN(2451544),
        PARTITION P3 VALUES LESS THAN(2451910),
        PARTITION P4 VALUES LESS THAN(2452275),
        PARTITION P5 VALUES LESS THAN(2452640),
        PARTITION P6 VALUES LESS THAN(2453005),
        PARTITION P7 VALUES LESS THAN(MAXVALUE)
);

-- Create an index named ds_inventory_p1_index1.
openGauss=# CREATE INDEX ds_inventory_p1_index1 ON tpcds.inventory_p1 (INV_ITEM_SK) LOCAL;

-- Cluster the tpcds.inventory_p1 table.
openGauss=# CLUSTER tpcds.inventory_p1 USING ds_inventory_p1_index1;

-- Cluster the p3 partition.
openGauss=# CLUSTER tpcds.inventory_p1 PARTITION (p3) USING ds_inventory_p1_index1;

-- Cluster the tables that can be clustered in the database.
openGauss=# CLUSTER;

-- Delete the index.
openGauss=# DROP INDEX tpcds.ds_inventory_p1_index1;

-- Delete the partitioned table.
openGauss=# DROP TABLE tpcds.inventory_p1;
```

Suggestions

- cluster
  - It is recommended that you run **ANALYZE** on a newly clustered table. Otherwise, the optimizer might make poor choices of query plans.
  - **CLUSTER** cannot be executed in transactions.

11.14.50 COMMENT

Function

COMMENT defines or changes the comment of an object.

Precautions

- Each object stores only one comment. Therefore, you need to modify a comment and issue a new **COMMENT** command to the same object. To delete the comment, write **NULL** at the position of the text string. When an object is deleted, the comment is automatically deleted.

- Currently, there is no security protection for viewing comments. Any user connected to a database can view all the comments for objects in the database. For shared objects such as databases, roles, and tablespaces, comments are stored globally so any user connected to any database in the cluster can see all the comments for shared objects. Therefore, do not put security-critical information in comments.
- To comment objects, you must be an object owner or user granted the COMMENT permission. The system administrator has this permission by default.
- Roles do not have owners, so the rule for **COMMENT ON ROLE** is that you must be an administrator to comment on an administrator role, or have the **CREATEROLE** permission to comment on non-administrator roles. A system administrator can comment on all objects.

Syntax

```
COMMENT ON
{
  AGGREGATE agg_name (agg_type [, ...] ) |
  CAST (source_type AS target_type) |
  COLLATION object_name |
  COLUMN { table_name.column_name | view_name.column_name } |
  CONSTRAINT constraint_name ON table_name |
  CONVERSION object_name |
  DATABASE object_name |
  DOMAIN object_name |
  EXTENSION object_name |
  FOREIGN DATA WRAPPER object_name |
  FOREIGN TABLE object_name |
  FUNCTION function_name ( [ {[ argname ] [ argmode ] argtype} [, ...] ] ) |
  INDEX object_name |
  LARGE OBJECT large_object_oid |
  OPERATOR operator_name (left_type, right_type) |
  OPERATOR CLASS object_name USING index_method |
  OPERATOR FAMILY object_name USING index_method |
  [ PROCEDURAL ] LANGUAGE object_name |
  ROLE object_name |
  SCHEMA object_name |
  SERVER object_name |
  TABLE object_name |
  TABLESPACE object_name |
  TEXT SEARCH CONFIGURATION object_name |
  TEXT SEARCH DICTIONARY object_name |
  TEXT SEARCH PARSER object_name |
  TEXT SEARCH TEMPLATE object_name |
  TYPE object_name |
  VIEW object_name |
  TRIGGER trigger_name ON table_name
}
   IS 'text';
```

Parameter Description

- **agg_name**

  Specifies the new name of an aggregate function.
- **agg_type**

  Specifies the data type of the aggregate function parameters.
- **source_type**

  Specifies the source data type of the cast.

- **target_type**

  Specifies the target data type of the cast.
- **object_name**

  Specifies the name of an object.
- **table_name.column_name**

  view_name.column_name

  Specifies the column whose comment is defined or modified. You can add the table name or view name as the prefix.
- **constraint_name**

  Specifies the table constraint whose comment is defined or modified.
- **table_name**

  Specifies the name of a table.
- **function_name**

  Specifies the function whose comment is defined or modified.
- **argname,argmode,argtype**

  Specifies the name, schema, and type of the function parameters.
- **large_object_oid**

  Specifies the OID of the large object whose comment is defined or modified.
- **operator_name**

  Specifies the name of the operator.
- **left_type,right_type**

  Specifies the data type of the operator parameters (optionally schema-qualified). If the prefix or suffix operator does not exist, the **NONE** option can be added.
- **trigger_name**

  Specifies the trigger name.
- **text**

  Specifies the comment content.

Examples

```
openGauss=# CREATE TABLE tpcds.customer_demographics_t2
(
    CD_DEMO_SK                INTEGER               NOT NULL,
    CD_GENDER                 CHAR(1)                       ,
    CD_MARITAL_STATUS         CHAR(1)                       ,
    CD_EDUCATION_STATUS       CHAR(20)                      ,
    CD_PURCHASE_ESTIMATE      INTEGER                       ,
    CD_CREDIT_RATING          CHAR(10)                      ,
    CD_DEP_COUNT              INTEGER                       ,
    CD_DEP_EMPLOYED_COUNT     INTEGER                       ,
    CD_DEP_COLLEGE_COUNT      INTEGER
)
WITH (ORIENTATION = COLUMN,COMPRESSION=MIDDLE)
;

-- Comment out the tpcds.customer_demographics_t2.cd_demo_sk column.
openGauss=# COMMENT ON COLUMN tpcds.customer_demographics_t2.cd_demo_sk IS 'Primary key of customer demographics table.';

-- Create a view consisting of rows with c_customer_sk less than 150.
```

```
openGauss=# CREATE VIEW tpcds.customer_details_view_v2 AS
    SELECT *
    FROM tpcds.customer
    WHERE c_customer_sk < 150;

-- Comment out the tpcds.customer_details_view_v2 view.
openGauss=# COMMENT ON VIEW tpcds.customer_details_view_v2 IS 'View of customer detail';

-- Delete the view.
openGauss=# DROP VIEW tpcds.customer_details_view_v2;

-- Delete the tpcds.customer_demographics_t2 table.
openGauss=# DROP TABLE tpcds.customer_demographics_t2;
```

11.14.51 COMMIT | END

Function

COMMIT or **END** commits all operations of a transaction.

Precautions

Only the creator of a transaction or the system administrator can run the **COMMIT** command. The creation and commit operations do not need to be in different sessions.

Syntax

```
{ COMMIT | END } [ WORK | TRANSACTION ] ;
```

Parameter Description

- **COMMIT | END**

  Commits the current transaction and makes all changes made by the transaction become visible to others.

- **WORK | TRANSACTION**

  Specifies an optional keyword, which has no effect except increasing readability.

Examples

```
-- Create a table.
openGauss=# CREATE TABLE tpcds.customer_demographics_t2
(
    CD_DEMO_SK                INTEGER               NOT NULL,
    CD_GENDER                 CHAR(1)                      ,
    CD_MARITAL_STATUS         CHAR(1)                      ,
    CD_EDUCATION_STATUS       CHAR(20)                     ,
    CD_PURCHASE_ESTIMATE      INTEGER                      ,
    CD_CREDIT_RATING          CHAR(10)                     ,
    CD_DEP_COUNT              INTEGER                      ,
    CD_DEP_EMPLOYED_COUNT     INTEGER                      ,
    CD_DEP_COLLEGE_COUNT      INTEGER
)
WITH (ORIENTATION = COLUMN,COMPRESSION=MIDDLE)
;

-- Start a transaction.
openGauss=# START TRANSACTION;

-- Insert data.
```

```
openGauss=# INSERT INTO tpcds.customer_demographics_t2 VALUES(1,'M', 'U', 'DOCTOR DEGREE', 1200, 
'GOOD', 1, 0, 0);
openGauss=# INSERT INTO tpcds.customer_demographics_t2 VALUES(2,'F', 'U', 'MASTER DEGREE', 300, 
'BAD', 1, 0, 0);

-- Commit the transaction to make all changes permanent.
openGauss=# COMMIT;

-- Query data.
openGauss=# SELECT * FROM tpcds.customer_demographics_t2;

-- Delete the tpcds.customer_demographics_t2 table.
openGauss=# DROP TABLE tpcds.customer_demographics_t2;
```

Helpful Links

ROLLBACK

11.14.52 COMMIT PREPARED

Function

COMMIT PREPARED commits a prepared two-phase transaction.

Precautions

- The function is only available in maintenance mode (when the GUC parameter **xc_maintenance_mode** is **on**). Exercise caution when enabling the mode. It is used by maintenance engineers for troubleshooting. Common users should not use the mode.
- Only the creator of a transaction or the system administrator can run the **COMMIT PREPARED** command. The creation and commit operations do not need to be in different sessions.
- The transaction function is maintained automatically by the database, and should be not visible to users.

Syntax

```
COMMIT PREPARED transaction_id ;
COMMIT PREPARED transaction_id WITH CSN;
```

Parameter Description

- **transaction_id**

  Specifies the identifier of the transaction to be committed. The identifier must be different from those for current prepared transactions.
- **CSN (commit sequence number)**

  Specifies the sequence number of the transaction to be committed. It is a 64-bit, incremental, unsigned number.

Examples

```
COMMIT PREPARED commits a transaction whose identifier is trans_test.
openGauss=# COMMIT PREPARED 'trans_test';
```

Helpful Links

PREPARE TRANSACTION and **ROLLBACK PREPARED**

11.14.53 COPY

Function

COPY copies data between tables and files.

COPY FROM copies data from a file to a table, and **COPY TO** copies data from a table to a file.

Precautions

- When the **enable_copy_server_files** parameter is disabled, only the initial user is allowed to run the **COPY FROM FILENAME** or **COPY TO FILENAME** statement. When the **enable_copy_server_files** parameter is enabled, users with the SYSADMIN permission or users who inherit the gs_role_copy_files permission of the built-in role are allowed to run the **COPY FROM FILENAME** or **COPY TO FILENAME** statement. By default, **COPY FROM FILENAME** or **COPY TO FILENAME** cannot be run for database configuration file, key files, certificate files, and audit logs to prevent unauthorized users from viewing or modifying sensitive files. When **enable_copy_server_files** is set to **on**, the administrator can use the GUC parameter **safe_data_path** to set the path for common users to import and export to the subpath of the set path. If this GUC parameter is not set (by default), the path used by common users is not blocked.
- **COPY** applies only to tables but not views.
- **COPY TO** requires the select permission on the table to be read, and **COPY FROM** requires the insert permission on the table to be inserted.
- If a list of columns is specified, **COPY** copies only the data of the specified columns between the file and the table. If a table has any columns that are not in the column list, **COPY FROM** inserts default values for those columns.
- If a data source file is specified, the server must be able to access the file. If **STDIN** is specified, data flows between the client and the server. When entering data, use the **TAB** key to separate the columns of the table and use a backslash and a period (\.) in a new row to indicate the end of the input.
- **COPY FROM** throws an error if any row in the data file contains more or fewer columns than expected.
- The end of the data can be represented by a line that contains only backslashes and periods (\.). If data is read from a file, the end flag is unnecessary. If data is copied between client applications, an end tag must be provided.
- In **COPY FROM**, **\N** is an empty string. To enter the actual value **\N**, use **\\N**.
- **COPY FROM** does not support data preprocessing during data import, such as expression operation and default value filling. If you need to preprocess data during the import, you need to import the data to a temporary table and then run SQL statements to insert the data to the table through operations. However, this method causes I/O expansion and reduces the import performance.

- When a data format error occurs during **COPY FROM** execution, the transaction is rolled back. However, the error information is insufficient, making it difficult to locate the error data from a large amount of raw data.
- **COPY FROM** and **COPY TO** apply to low concurrency and local import and export of a small amount of data.
- If the target table has triggers, **COPY** is supported.
- Ensure that the generated column is not in the list of the specified column in the **COPY** statement. When **COPY... TO** is used to export data, if no column list is specified, all columns except the generated columns in the table are exported. When **COPY... FROM** is used to import data, the generated columns are automatically updated and saved as common columns.

Syntax

- Copy data from a file to a table.

```
COPY table_name [ ( column_name [, ...] ) ]
    FROM { 'filename' | STDIN }
    [ [ USING ] DELIMITERS 'delimiters' ]
    [ WITHOUT ESCAPING ]
    [ LOG ERRORS ]
    [ REJECT LIMIT 'limit' ]
    [ WITH ( option [, ...] ) ]
    | copy_option
    | TRANSFORM  ( { column_name [ data_type ] [ AS transform_expr ] } [, ...] )
    | FIXED FORMATTER ( { column_name( offset, length ) } [, ...] ) [ ( option [, ...] ) | copy_option
[  ...] ] ];
```

NOTE

In the syntax, **FIXED FORMATTER ({ column_name(offset, length) } [, ...])** and non-conflicting items of **[copy_option [...]]** can be in any sequence.

- Copy data from a table to a file.

```
COPY table_name [ ( column_name [, ...] ) ]
    TO { 'filename' | STDOUT }
    [ [ USING ] DELIMITERS 'delimiters' ]
    [ WITHOUT ESCAPING ]
    [ WITH ( option [, ...] ) ]
    | copy_option
    | FIXED FORMATTER ( { column_name( offset, length ) } [, ...] ) [ ( option [, ...] ) | copy_option
[  ...] ] ];

COPY query
    TO { 'filename' | STDOUT }
    [ WITHOUT ESCAPING ]
    [ WITH ( option [, ...] ) ]
    | copy_option
    | FIXED FORMATTER ( { column_name( offset, length ) } [, ...] ) [ ( option [, ...] ) | copy_option
[  ...] ] ];
```

NOTE

1. The syntax constraints of **COPY TO** are as follows:
   (query) is incompatible with **[USING] DELIMITER**. If the data comes from a query result, **COPY TO** cannot specify **[USING] DELIMITERS**.
2. Use spaces to separate **copy_option** following **FIXED FORMATTTER**.
3. **copy_option** is the native parameter, while **option** is the parameter imported by a compatible foreign table.
4. In the syntax, **FIXED FORMATTER ({ column_name(offset, length) } [, ...])** and non-conflicting items of **[copy_option [...]]** can be in any sequence.

The syntax of the optional parameter **option** is as follows:

```
FORMAT 'format_name'
| OIDS [ boolean ]
| DELIMITER 'delimiter_character'
| NULL 'null_string'
| HEADER [ boolean ]
| FILEHEADER 'header_file_string'
| FREEZE [ boolean ]
| QUOTE 'quote_character'
| ESCAPE 'escape_character'
| EOL 'newline_character'
| NOESCAPING [ boolean ]
| FORCE_QUOTE { ( column_name [, ...] ) | * }
| FORCE_NOT_NULL ( column_name [, ...] )
| ENCODING 'encoding_name'
| IGNORE_EXTRA_DATA [ boolean ]
| FILL_MISSING_FIELDS [ boolean ]
| COMPATIBLE_ILLEGAL_CHARS [ boolean ]
| DATE_FORMAT 'date_format_string'
| TIME_FORMAT 'time_format_string'
| TIMESTAMP_FORMAT 'timestamp_format_string'
| SMALLDATETIME_FORMAT 'smalldatetime_format_string'
```

The syntax of the optional parameter **copy_option** is as follows:

```
OIDS
| NULL 'null_string'
| HEADER
| FILEHEADER 'header_file_string'
| FREEZE
| FORCE_NOT_NULL column_name [, ...]
| FORCE_QUOTE { column_name [, ...] | * }
| BINARY
| CSV
| QUOTE [ AS ] 'quote_character'
| ESCAPE [ AS ] 'escape_character'
| EOL 'newline_character'
| ENCODING 'encoding_name'
| IGNORE_EXTRA_DATA
| FILL_MISSING_FIELDS [ { 'one' | 'multi' } ]
| COMPATIBLE_ILLEGAL_CHARS
| DATE_FORMAT 'date_format_string'
| TIME_FORMAT 'time_format_string'
| TIMESTAMP_FORMAT 'timestamp_format_string'
| SMALLDATETIME_FORMAT 'smalldatetime_format_string'
| SKIP int_number
| WHEN { ( start - end ) | column_name } { = | != } 'string'
| SEQUENCE ( { column_name ( integer [, incr] ) [, ...] } )
| FILLER ( { column_name [, ...] } )
| CONSTANT ( { column_name 'constant_string' [, ...] } )
```

Parameter Description

- **query**

  Specifies that the results are to be copied.

  Valid value: a **SELECT** or **VALUES** command in parentheses

- **table_name**

  Specifies the name (possibly schema-qualified) of an existing table.

  Value range: an existing table name

- **column_name**

  Specifies an optional list of columns to be copied.

  Value range: any columns. All columns will be copied if no column list is specified.

- **STDIN**

Specifies that input comes from the standard input.

- **STDOUT**

  Specifies that output goes to the standard output.

- **FIXED**

  Fixes column length. When the column length is fixed, **DELIMITER**, **NULL**, and **CSV** cannot be specified. When **FIXED** is specified, **BINARY**, **CSV**, and **TEXT** cannot be specified by **option** or **copy_option**.

  NOTE

  The definition of fixed length is as follows:

  1. The column length of each record is the same.
  2. Spaces are used for column padding. Columns of the numeric type are left-aligned and columns of the string type are right-aligned.
  3. No delimiters are used between columns.

- **[USING] DELIMITERS 'delimiters'**

  The string that separates columns within each row (line) of the file, and it cannot be larger than 10 bytes.

  Value range: The delimiter cannot include any of the following characters: \.abcdefghijklmnopqrstuvwxyz0123456789

  Value range: The default value is a tab character in text format and a comma in CSV format.

- **WITHOUT ESCAPING**

  Specifies, in text format, whether to escape the backslash (\) and its following characters.

  Value range: text only

- **LOG ERRORS**

  If this parameter is specified, the error tolerance mechanism for data type errors in the **COPY FROM** statement is enabled.

  Value range: a value set while data is imported using **COPY FROM**.

  NOTE

  The restrictions of this error tolerance parameter are as follows:

  - This error tolerance mechanism captures only the data type errors (DATA_EXCEPTION) that occur during data parsing of **COPY FROM** on the primary node of the database.
  - If existing error tolerance parameters (for example, **IGNORE_EXTRA_DATA**) of the **COPY** statement are enabled, the error of the corresponding type will be processed as specified by the parameters and no error will be reported. Therefore, the error table does not contain such error data.

- **LOG ERRORS DATA**

  The differences between **LOG ERRORS DATA** and **LOG ERRORS** are as follows:

  a. **LOG ERRORS DATA** fills the **rawrecord** column in the error tolerance table.

  b. Only users with the super permission can use the **LOG ERRORS DATA** parameter.

CAUTION

If error content is too complex, it may fail to be written to the error tolerance table by using **LOG ERRORS DATA**, causing the task failure.

For errors that cannot be read in certain code, the error codes are ERRCODE_CHARACTER_NOT_IN_REPERTOIRE and ERRCODE_UNTRANSLATABLE_CHARACTER. The rawrecord column is not recorded.

- **REJECT LIMIT 'limit'**

  Used with the **LOG ERROR** parameter to set the upper limit of the tolerated errors in the **COPY FROM** statement. If the number of errors exceeds the limit, later errors will be reported based on the original mechanism.

  Value range: a positive integer (1 to *INTMAX*) or **unlimited**

  Default value: If **LOG ERRORS** is not specified, an error will be reported. If **LOG ERRORS** is specified, the default value is **0**.

  NOTE

  In the error tolerance mechanism described in the description of **LOG ERRORS**, the count of **REJECT LIMIT** is calculated based on the number of data parsing errors on the primary node of the database where the **COPY FROM** statement is executed, not based on the number of all errors on the primary node.

- **FORMATTER**

  Defines the place of each column in the data file in fixed length mode. Defines the place of each column in the data file in the **column(***offset*,*length***)** format.

  Value range:

  - The value of **offset** must be larger than 0. The unit is byte.
  - The value of **length** must be larger than 0. The unit is byte.

  The total length of all columns must be less than 1 GB.

  Replace columns that are not in the file with null.

- **OPTION { option_name ' value ' }**

  Specifies all types of parameters of a compatible foreign table.

  - FORMAT

    Specifies the format of the source data file in the foreign table.

    Value range: **CSV**, **TEXT**, **FIXED**, and **BINARY**

    - The CSV file can process newline characters efficiently, but cannot process certain special characters well.
    - The TEXT file can process certain special characters efficiently, but cannot process newline characters well.
    - In FIXED files, the column length of each record is the same. Spaces are used for padding, and the excessive part will be truncated.
    - All data in the BINARY file is stored/read as binary format rather than as text. It is faster than the text and CSV formats, but a binary-format file is less portable.

Default value: **TEXT**

– DELIMITER

Specifies the character that separates columns within each row (line) of the file.

NOTE

- The value of **delimiter** cannot be **\r** or **\n**.
- A delimiter cannot be the same as the null value. The delimiter for the CSV format cannot be same as the **quote** value.
- The delimiter for the TEXT format data cannot contain lowercase letters, digits, or special characters (.\).
- The data length of a single row should be less than 1 GB. A row that has many columns using long delimiters cannot contain much valid data.
- You are advised to use multi-character delimiters or invisible delimiters. For example, you can use multi-characters (such as $^&) and invisible characters (such as 0x07, 0x08, and 0x1b).

Value range: a multi-character delimiter within 10 bytes

Default value:

- A tab character in text format
- A comma (,) in CSV format
- No delimiter in FIXED format

– NULL

Specifies the string that represents a null value.

Value range:

- A null value cannot be **\r** or **\n**. The maximum length is 100 characters.
- A null value cannot be the same as the **delimiter** or **quote** value.

Default value:

- The default value for the CSV format is an empty string without quotation marks.
- The default value for the TEXT format is **\N**.

– HEADER

Specifies whether a file contains a header with the names of each column in the file. **header** is available only for CSV and FIXED files.

When data is imported, if **header** is **on**, the first row of the data file will be identified as the header and ignored. If **header** is **off**, the first row will be identified as a data row.

When data is exported, if header is **on**, **fileheader** must be specified. If **header** is **off**, an exported file does not contain a header.

Value range: **true**, **on**, **false**, and **off**.

Default value: **false**

– QUOTE

Specifies a quoted character string for a CSV file.

Default value: single quotation marks ('')

NOTE

- The value of **quote** cannot be the same as that of the **delimiter** or **null** parameter.
- The value of **quote** must be a single-byte character.
- You are advised to set **quote** to an invisible character, such as **0x07**, **0x08**, or **0x1b**.

– ESCAPE

Specifies an escape character for a CSV file. The value must be a single-byte character.

Default value: single quotation marks ('') If the value is the same as that of **quote**, it will be replaced by **\0**.

– EOL 'newline_character'

Specifies the newline character style of the imported or exported data file.

Value range: multi-character newline characters within 10 bytes. Common newline characters include **\r** (0x0D), **\n** (0x0A), and **\r\n** (0x0D0A). Special newline characters include **$** and **#**.

NOTE

- The **EOL** parameter supports only the TEXT format for data import and export and does not support the CSV or FIXED format for data import. For forward compatibility, the EOL parameter can be set to **0x0D** or **0x0D0A** for data export in the CSV or FIXED format.
- The value of **EOL** cannot be the same as that of the **delimiter** or **null** parameter.
- The EOL parameter value cannot contain the following characters: .abcdefghijklmnopqrstuvwxyz0123456789.

– FORCE_QUOTE { (column_name [, ...]) | * }

In **CSV COPY TO** mode, forces quotation marks to be used for all non-null values in each specified column. Null values are not quoted.

Value range: an existing column name

– FORCE_NOT_NULL (column_name [, ...])

In **CSV COPY FROM** mode, the value for a specified column cannot be null.

Value range: an existing column name

– ENCODING

Specifies the encoding of data files. If this option is omitted, the current client encoding is used.

– IGNORE_EXTRA_DATA

Specifies whether to ignore excessive columns when the number of data source files exceeds the number of foreign table columns. This parameter is used only during data import.

Value range: **true**, **on**, **false**, and **off**.

  - If this parameter is set to **true** or **on** and the number of source data files exceeds the number of foreign table columns, excessive columns will be ignored.
  - When the parameter is **false** or **off**, and the number of data source files is more than the number of foreign table columns, the following error information will be displayed:

    ```
    extra data after last expected column
    ```

  Default value: **false**

  NOTICE

  If a newline character at the end of a row is missing and the row and another row are integrated into one, data in another row is ignored after the parameter is set to **true**.

- COMPATIBLE_ILLEGAL_CHARS

  Specifies whether to tolerate invalid characters during data import. The parameter is valid only for data import using **COPY FROM**.

  Value range: **true**, **on**, **false**, and **off**.

  - If this parameter is set to **true** or **on**, invalid characters are tolerated and imported to the database after conversion.
  - If this parameter is set to **false** or **off** and an error occurs when there are invalid characters, the import will be interrupted.

  Default value: **false** or **off**

  NOTE

  The rules for converting invalid characters are as follows:

  1. **\0** is converted to a space.

  2. Other invalid characters are converted to question marks.

  (3) If **compatible_illegal_chars** is set to **true** or **on**, invalid characters are tolerated. If **NULL**, **DELIMITER**, **QUOTE**, and **ESCAPE** are set to a spaces or question marks, errors like "illegal chars conversion may confuse COPY escape 0x20" will be displayed to prompt users to change parameter values that cause confusion, preventing import errors.

- FILL_MISSING_FIELDS

  Specifies how to handle the problem that the last column of a row in a source data file is lost during data import.

  Value range: **true**, **on**, **false**, and **off**.

  Default value: **false** or **off**

- DATE_FORMAT

  Specifies the DATE format for data import. The BINARY format is not supported. When data of such format is imported, error "cannot specify bulkload compatibility options in BINARY mode" will occur. The parameter is valid only for data import using **COPY FROM**.

  Value range: a valid DATE value For details, see **Date and Time Processing Functions and Operators**.

NOTE

You can use the **TIMESTAMP_FORMAT** parameter to set the DATE format to **TIMESTAMP** for data import. For details, see **TIMESTAMP_FORMAT** below.

- TIME_FORMAT

  Specifies the TIME format for data import. The BINARY format is not supported. When data of such format is imported, error "cannot specify bulkload compatibility options in BINARY mode" will occur. The parameter is valid only for data import using **COPY FROM**.

  Value range: a valid TIME value. Time zones cannot be used. For details, see **Date and Time Processing Functions and Operators**.

- TIMESTAMP_FORMAT

  Specifies the TIMESTAMP format for data import. The BINARY format is not supported. When data of such format is imported, error "cannot specify bulkload compatibility options in BINARY mode" will occur. The parameter is valid only for data import using **COPY FROM**.

  Value range: a valid TIMESTAMP value. Time zones cannot be used. For details, see **Date and Time Processing Functions and Operators**.

- SMALLDATETIME_FORMAT

  Specifies the SMALLDATETIME format for data import. The BINARY format is not supported. When data of such format is imported, error "cannot specify bulkload compatibility options in BINARY mode" will occur. The parameter is valid only for data import using **COPY FROM**.

  Value range: a valid SMALLDATETIME value. For details, see **Date and Time Processing Functions and Operators**.

- **COPY_OPTION { option_name ' value ' }**

  Specifies all types of native parameters of **COPY**.

  - NULL null_string

    Specifies the string that represents a null value.

    NOTICE

    When using **COPY FROM**, any data item that matches this string will be stored as a null value, so make sure that you use the same string as you used with **COPY TO**.

    Value range:

    - A null value cannot be **\r** or **\n**. The maximum length is 100 characters.
    - A null value cannot be the same as the **delimiter** or **quote** value.

    Default value:

    - The default value for the TEXT format is **\N**.
    - The default value for the CSV format is an empty string without quotation marks.

– HEADER

  Specifies whether a file contains a header with the names of each column in the file. **header** is available only for CSV and FIXED files.

  When data is imported, if **header** is **on**, the first row of the data file will be identified as the header and ignored. If **header** is **off**, the first row will be identified as a data row.

  When data is exported, if header is **on**, **fileheader** must be specified. If **header** is **off**, an exported file does not contain a header.

– FILEHEADER

  Specifies a file that defines the content in the header for exported data. The file contains data description of each column.

  NOTICE

  - This parameter is available only when **header** is **on** or **true**.
  - **fileheader** specifies an absolute path.
  - The file can contain only one row of header information, and ends with a newline character. Excess rows will be discarded. (Header information cannot contain newline characters.)
  - The length of the file including the newline character cannot exceed 1 MB.

– FREEZE

  Sets the **COPY** loaded data row as **frozen**, like these data have executed **VACUUM FREEZE**.

  This is a performance option of initial data loading. The data will be frozen only when the following three requirements are met:

  - The table being loaded has been created or truncated in the same transaction before copying.
  - There are no cursors open in the current transaction.
  - There are no original snapshots in the current transaction.

  NOTE

  When **COPY** is completed, all the other sessions will see the data immediately. However, this violates the general principle of MVCC visibility, and users should understand that this may cause potential risks.

– FORCE NOT NULL column_name [, ...]

  In **CSV COPY FROM** mode, the specified column is not null. If the column is null, its value is regarded as a string of 0 characters.

  Value range: an existing column name

– FORCE QUOTE { column_name [, ...] | * }

  In **CSV COPY TO** mode, forces quotation marks to be used for all non-null values in each specified column. Null values are not quoted.

  Value range: an existing column name

– BINARY

  Specifies that data is stored and read in binary mode instead of text mode. In binary mode, you cannot declare **DELIMITER**, **NULL**, or **CSV**. When **BINARY** is specified, **CSV**, **FIXED**, and **TEXT** cannot be specified through **option** or **copy_option**.

– CSV

  Enables the CSV mode. When **CSV** is specified, **BINARY**, **FIXED**, and **TEXT** cannot be specified through **option** or **copy_option**.

– QUOTE [AS] 'quote_character'

  Specifies a quoted character string for a CSV file.

  Default value: single quotation marks ('')

  NOTE

  - The value of **quote** cannot be the same as that of the **delimiter** or **null** parameter.
  - The value of **quote** must be a single-byte character.
  - You are advised to set **quote** to an invisible character, such as **0x07**, **0x08**, or **0x1b**.

– ESCAPE [AS] 'escape_character'

  Specifies an escape character for a CSV file. The value must be a single-byte character.

  The default value is single quotation marks (''). If the value is the same as that of **quote**, it will be replaced by **\0**.

– EOL 'newline_character'

  Specifies the newline character style of the imported or exported data file.

  Value range: multi-character newline characters within 10 bytes. Common newline characters include **\r** (0x0D), **\n** (0x0A), and **\r\n** (0x0D0A). Special newline characters include **$** and **#**.

  NOTE

  - The **EOL** parameter supports only the TEXT format for data import and export and does not support the CSV or FIXED format. For forward compatibility, the **EOL** parameter can be set to **0x0D** or **0x0D0A** for data export in the CSV or FIXED format.
  - The value of **EOL** cannot be the same as that of the **delimiter** or **null** parameter.
  - The EOL parameter value cannot contain the following characters: .abcdefghijklmnopqrstuvwxyz0123456789.

– ENCODING 'encoding_name'

  Specifies the name of a file encoding format.

  Value range: a valid encoding format

  Default value: current encoding format

– IGNORE_EXTRA_DATA

  Specifies that when the number of data source files exceeds the number of foreign table columns, excess columns at the end of the row are ignored. This parameter is used only during data import.

If this parameter is not used and the number of columns in the data source file is greater than that defined in the foreign table, the following error information is displayed:

```
extra data after last expected column
```

- COMPATIBLE_ILLEGAL_CHARS

  Specifies that invalid characters are tolerated during data import. Invalid characters are converted and then imported to the database. No error is reported and the import is not interrupted. The BINARY format is not supported. When data of such format is imported, error "cannot specify bulkload compatibility options in BINARY mode" will occur. The parameter is valid only for data import using **COPY FROM**.

  If this parameter is not used, an error is reported when invalid characters are encountered during the import, and the import is interrupted.

  NOTE

  The rules for converting invalid characters are as follows:

  1. **\0** is converted to a space.

  2. Other invalid characters are converted to question marks.

  3. When **compatible_illegal_chars** is set to **true** or **on**, after invalid characters such as **NULL**, **DELIMITER**, **QUOTE**, and **ESCAPE** are converted to spaces or question marks, an error message like "illegal chars conversion may confuse COPY escape 0x20" will be displayed to remind you of possible parameter confusion caused by the conversion.

- FILL_MISSING_FIELDS [{ 'one' | 'multi' }]

  Specifies how to handle the problem that the last columns of a row in a source data file are lost during data import. If **one** or **multi** is not specified or **one** is specified, the missing of the last column is handled in the default mode. If **multi** is specified, the missing of the last multiple columns are handled in the default mode.

  Value range: **true**, **on**, **false**, and **off**.

  Default value: **false** or **off**

  NOTICE

  Do not specify this option. Currently, it does not enable error tolerance, but will make the parser ignore the said errors during data parsing on the primary node of the database. Such errors will not be recorded in the COPY error table (enabled using **LOG ERRORS REJECT LIMIT**) but will be reported later by database node. Therefore, do not specify this option.

- DATE_FORMAT 'date_format_string'

  Specifies the DATE format for data import. The BINARY format is not supported. When data of such format is imported, error "cannot specify bulkload compatibility options in BINARY mode" will occur. The parameter is valid only for data import using **COPY FROM**.

  Value range: a valid DATE value For details, see **Date and Time Processing Functions and Operators**.

NOTE

You can use the **TIMESTAMP_FORMAT** parameter to set the DATE format to **TIMESTAMP** for data import. For details, see **TIMESTAMP_FORMAT** below.

- TIME_FORMAT 'time_format_string'

  Specifies the TIME format for data import. The BINARY format is not supported. When data of such format is imported, error "cannot specify bulkload compatibility options in BINARY mode" will occur. The parameter is valid only for data import using **COPY FROM**.

  Value range: a valid TIME value. Time zones cannot be used. For details, see **Date and Time Processing Functions and Operators**.

- TIMESTAMP_FORMAT 'timestamp_format_string'

  Specifies the TIMESTAMP format for data import. The BINARY format is not supported. When data of such format is imported, error "cannot specify bulkload compatibility options in BINARY mode" will occur. The parameter is valid only for data import using **COPY FROM**.

  Value range: a valid TIMESTAMP value. Time zones cannot be used. For details, see **Date and Time Processing Functions and Operators**.

- SMALLDATETIME_FORMAT 'smalldatetime_format_string'

  Specifies the SMALLDATETIME format for data import. The BINARY format is not supported. When data of such format is imported, error "cannot specify bulkload compatibility options in BINARY mode" will occur. The parameter is valid only for data import using **COPY FROM**.

  Value range: a valid SMALLDATETIME value. For details, see **Date and Time Processing Functions and Operators**.

- TRANSFORM ({ column_name [data_type] [AS transform_expr] } [, ...])

  Specify the conversion expression of each column in the table. **data_type** specifies the data type of the column in the expression parameter. **transform_expr** is the target expression that returns the result value whose data type is the same as that of the target column in the table. For details about the expression, see **Expressions**.

- SKIP int_number

  Specifies that the first *int_number* rows of the data file are skipped during data import.

- WHEN { (start - end) | column_name } { = | != } 'string'

  When data is imported, each row of data is checked. Only the rows that meet the WHEN condition are imported to the table.

- SEQUENCE ({ column_name (integer [, incr]) [, ...] })

  During data import, columns modified by SEQUENCE do not read data from the data file. The values are incremented by the value of **incr** based on the specified integer. If **incr** is not specified, the values are incremented from 1 by default.

- FILLER ({ column_name [, ...] })

  When data is imported, the column modified by FILLER is discarded after being read from the data file.

NOTE

To use FILLER, you need to specify the list of columns to be copied. During data processing, data is processed based on the position of the **filler** column in the column list.

- CONSTANT ({ column_name 'constant_string' [, ...] })

  When data is imported, the column modified by CONSTANT is not read from the data file, and **constant_string** is used to assign a value to the column.

The following special backslash sequences are recognized by **COPY FROM**:

- **\b**: Backslash (ASCII 8)
- **\f**: Form feed (ASCII 12)
- **\n**: Newline character (ASCII 10)
- **\r**: Carriage return character (ASCII 13)
- **\t**: Tab (ASCII 9)
- **\v**: Vertical tab (ASCII 11)
- **\digits**: Backslash followed by one to three octal digits specifies that the ASCII value is the character with that numeric code.
- **\xdigits**: Backslash followed by an x and one or two hex digits specifies the character with that numeric code.

Permission Control Examples

```
openGauss=> copy t1 from '/home/xy/t1.csv';
ERROR:  COPY to or from a file is prohibited for security concerns
HINT:  Anyone can COPY to stdout or from stdin. gsql's \copy command also works for anyone.
openGauss=> grant gs_role_copy_files to xxx;
```

This error occurs because a non-initial user does not have the COPY permission. To solve this problem, set the **enable_copy_server_files** parameter to **on**. Then, the administrator can use the COPY function. To perform the COPY operation, common users must also join the **gs_role_copy_files** group.

Examples

```
-- Copy data from the tpcds.ship_mode file to the /home/omm/ds_ship_mode.dat file:
openGauss=# COPY tpcds.ship_mode TO '/home/omm/ds_ship_mode.dat';

-- Output tpcds.ship_mode to stdout.
openGauss=# COPY tpcds.ship_mode TO stdout;

-- Create the tpcds.ship_mode_t1 table.
openGauss=# CREATE TABLE tpcds.ship_mode_t1
(
    SM_SHIP_MODE_SK           INTEGER               NOT NULL,
    SM_SHIP_MODE_ID           CHAR(16)              NOT NULL,
    SM_TYPE                   CHAR(30)                      ,
    SM_CODE                   CHAR(10)                      ,
    SM_CARRIER                CHAR(20)                      ,
    SM_CONTRACT               CHAR(20)
)
WITH (ORIENTATION = COLUMN,COMPRESSION=MIDDLE)
;

-- Copy data from stdin to the tpcds.ship_mode_t1 table.
openGauss=# COPY tpcds.ship_mode_t1 FROM stdin;
```

```
-- Copy data from the /home/omm/ds_ship_mode.dat file to the tpcds.ship_mode_t1 table.
openGauss=# COPY tpcds.ship_mode_t1 FROM '/home/omm/ds_ship_mode.dat';

-- Copy data from the /home/omm/ds_ship_mode.dat file to the tpcds.ship_mode_t1 table, convert the
data using the TRANSFORM expression, and insert the 10 characters on the left of the SM_TYPE column
into the table.
openGauss=# COPY tpcds.ship_mode_t1 FROM '/home/omm/ds_ship_mode.dat' TRANSFORM (SM_TYPE AS
LEFT(SM_TYPE, 10));

-- Copy data from the /home/omm/ds_ship_mode.dat file to the tpcds.ship_mode_t1 table, with the
import format set to TEXT (format 'text'), the delimiter set to \t' (delimiter E'\t'), excessive columns
ignored (ignore_extra_data 'true'), and characters not escaped (noescaping 'true').
openGauss=# COPY tpcds.ship_mode_t1 FROM '/home/omm/ds_ship_mode.dat' WITH(format 'text',
delimiter E'\t', ignore_extra_data 'true', noescaping 'true');

-- Copy data from the /home/omm/ds_ship_mode.dat file to the tpcds.ship_mode_t1 table, with the
import format set to FIXED, fixed-length format specified (FORMATTER(SM_SHIP_MODE_SK(0, 2),
SM_SHIP_MODE_ID(2,16), SM_TYPE(18,30), SM_CODE(50,10), SM_CARRIER(61,20),
SM_CONTRACT(82,20))), excessive columns ignored (ignore_extra_data), and headers included (header).
openGauss=# COPY tpcds.ship_mode_t1 FROM '/home/omm/ds_ship_mode.dat' FIXED
FORMATTER(SM_SHIP_MODE_SK(0, 2), SM_SHIP_MODE_ID(2,16), SM_TYPE(18,30), SM_CODE(50,10),
SM_CARRIER(61,20), SM_CONTRACT(82,20)) header ignore_extra_data;

-- Delete the tpcds.ship_mode_t1 table.
openGauss=# DROP TABLE tpcds.ship_mode_t1;
```

11.14.54 CREATE AGGREGATE

Function

CREATE AGGREGATE defines a new aggregate function.

Syntax

```
CREATE AGGREGATE name ( input_data_type [ , ... ] ) (
    SFUNC = sfunc,
    STYPE = state_data_type
    [ , FINALFUNC = ffunc ]
    [ , INITCOND = initial_condition ]
    [ , SORTOP = sort_operator ]
)

or the old syntax

CREATE AGGREGATE name (
    BASETYPE = base_type,
    SFUNC = sfunc,
    STYPE = state_data_type
    [ , FINALFUNC = ffunc ]
    [ , INITCOND = initial_condition ]
    [ , SORTOP = sort_operator ]
)
```

Parameter Description

- **name**

  Name (optionally schema-qualified) of the aggregate function to be created.
- **input_data_type**

  Data type of the input to be processed by the aggregate function. To create a zero-parameter aggregate function, you can use an asterisk (*) instead of a list of input data types. (count(*) is an instance of this aggregate function.)
- **base_type**

In the **CREATE AGGREGATE** syntax, the input data type is specified by the **basetype** parameter instead of following **name**. Note that the previous syntax allows only one input parameter. To create a zero-parameter aggregate function, you can set **basetype** to **ANY** instead of *.

- **sfunc**

  Name of the state conversion function that will be called on each input line. For an aggregate function with N parameters, **sfunc** must have more than one parameter. The first parameter is of the **state_data_type** type, and the other parameters match the declared input data types. The function must return a value of the **state_data_type** type. This function accepts the current state value and the current input data, and returns the next state value.

- **state_data_type**

  Data type of the aggregation status value.

- **ffunc**

  Final processing function called after all the input lines have been converted, which calculates the result of aggregation. This function must accept a parameter of **state_data_type**. The output data type of the aggregation is defined as the return type of this function. If **ffunc** is not specified, the state value of the aggregation result is used as the aggregation result, and the output type is **state_data_type**.

- **initial_condition**

  Initial setting (value) of a state value. It must be a text constant value acceptable to **state_data_type**. If not specified, the initial state value is **NULL**.

- **sort_operator**

  Sort operator used for MIN or MAX aggregation. This is just an operator name (optionally schema-qualified). This operator assumes that the input data type is the same as that of aggregation.

Examples

```
CREATE AGGREGATE sum (complex)
(
   sfunc = complex_add,
   stype = complex,
   initcond = '(0,0)'
);

SELECT sum(a) FROM test_complex;

  sum
-----------
 (34,53.9)
```

11.14.55 CREATE AUDIT POLICY

Function

CREATE AUDIT POLICY creates a unified audit policy.

Precautions

Only users with the **poladmin** or **sysadmin** permission, or the initial user can perform this operation.

The masking policy takes effect only after the security policy is enabled, that is, **enable_security_policy** is set to **on**. For details, see "Database Configuration > Database Security Management Policies > Unified Auditing" in *Security Hardening Guide*.

Syntax

```
CREATE AUDIT POLICY [ IF NOT EXISTS ] policy_name { { privilege_audit_clause | access_audit_clause }
[ filter_group_clause ] [ ENABLE | DISABLE ] };
```

- privilege_audit_clause
  ```
  PRIVILEGES { DDL | ALL } [ ON LABEL ( resource_label_name [, ... ] ) ]
  ```
- access_audit_clause
  ```
  ACCESS { DML | ALL } [ ON LABEL ( resource_label_name [, ... ] ) ]
  ```
- filter_group_clause
  ```
  FILTER ON { ( FILTER_TYPE ( filter_value [, ... ] ) ) [, ... ] }
  ```

Parameter Description

- **policy_name**

  Specifies the audit policy name, which must be unique.

  Value range: a string. It must comply with the naming convention.
- **DDL**

  Specifies the operations that are audited within the database: **CREATE**, **ALTER**, **DROP**, **ANALYZE**, **COMMENT**, **GRANT**, **REVOKE**, **SET**, **SHOW**, **LOGIN_ANY**, **LOGIN_FAILURE**, **LOGIN_SUCCESS**, and **LOGOUT**.
- **ALL**

  Indicates all operations supported by the specified DDL statements in the database.
- **resource_label_name**

  Specifies the resource label name.
- **DML**

  Specifies the operations that are audited within the database: **SELECT**, **COPY**, **DEALLOCATE**, **DELETE**, **EXECUTE**, **INSERT**, **PREPARE**, **REINDEX**, **TRUNCATE**, and **UPDATE**.
- **FILTER_TYPE**

  Specifies the types of information to be filtered by the audit, including **IP**, **APP**, and **ROLES**.
- **filter_value**

  Indicates the detailed information to be filtered.
- **ENABLE|DISABLE**

  Enables or disables the unified audit policy. If **ENABLE|DISABLE** is not specified, **ENABLE** is used by default.

Examples

```
-- Create users dev_audit and bob_audit.
openGauss=# CREATE USER dev_audit PASSWORD 'dev@1234';
CREATE USER bob_audit password 'bob@1234';

-- Create table tb_for_audit.
```

```
openGauss=# CREATE TABLE tb_for_audit(col1 text, col2 text, col3 text);

-- Create a resource label.
openGauss=# CREATE RESOURCE LABEL adt_lb0 add TABLE(tb_for_audit);

-- Perform the CREATE operation on the database to create an audit policy.
openGauss=# CREATE AUDIT POLICY adt1 PRIVILEGES CREATE;

-- Perform the SELECT operation on the database to create an audit policy.
openGauss=# CREATE AUDIT POLICY adt2 ACCESS SELECT;

-- Create an audit policy to audit only the CREATE operations performed on the adt_lb0 resource by users
dev_audit and bob_audit.
openGauss=# CREATE AUDIT POLICY adt3 PRIVILEGES CREATE ON LABEL(adt_lb0) FILTER ON
ROLES(dev_audit, bob_audit);

-- Create an audit policy to audit only the SELECT, INSERT, and DELETE operations performed on the
adt_lb0 resource by users dev_audit and bob_audit using client tools psql and gsql on the servers whose
IP addresses are 10.20.30.40 and 127.0.0.0/24.
openGauss=# CREATE AUDIT POLICY adt4 ACCESS SELECT ON LABEL(adt_lb0), INSERT ON LABEL(adt_lb0),
DELETE FILTER ON ROLES(dev_audit, bob_audit), APP(psql, gsql), IP('10.20.30.40', '127.0.0.0/24');
```

Helpful Links

ALTER AUDIT POLICY and DROP AUDIT POLICY

11.14.56 CREATE CAST

Function

CREATE CAST defines a conversion.

Syntax

```
CREATE CAST (source_type AS target_type)
    WITH FUNCTION function_name (argument_type [, ...])
    [ AS ASSIGNMENT | AS IMPLICIT ]

CREATE CAST (source_type AS target_type)
    WITHOUT FUNCTION
    [ AS ASSIGNMENT | AS IMPLICIT ]

CREATE CAST (source_type AS target_type)
    WITH INOUT
    [ AS ASSIGNMENT | AS IMPLICIT ]
```

Parameter Description

- **source_type**

  Type of the source data to be converted.

- **target_type**

  Type of the target data to be converted.

- **function_name(argument_type [, ...])**

  Function used for conversion. The function name can be modified with a schema name. If it is not modified with a schema name, the function will be found in the schema search path. The result data type of the function must match the target type of the conversion. Its parameters are discussed below.

- **WITHOUT FUNCTION**

Indicates that the source type is a binary castable to the target type, so no function is needed to perform this conversion.

- **WITH INOUT**

  Indicates that the conversion is an I/O conversion, which is performed by calling the output function of the source data type and transferring the result to the input function of the target data type.

- **AS ASSIGNMENT**

  Indicates that the conversion can be implicitly called in assignment mode.

- **AS IMPLICIT**

  Indicates that the transformation can be implicitly called in any environment.

  A conversion implementation function can have one to three parameters. The type of the first parameter must be the same as that of the source type to be converted, or can be forcibly converted from the binary of the source type to be converted. If the second parameter exists, it must be of the integer type. It receives these type modifiers associated with the target type, or **-1** if nothing is present. If the third parameter exists, it must be of the Boolean type. If the conversion is an explicit type conversion, **true** is received. Otherwise, **false** is received.

  The return type of a conversion function must be the same as the target type of the conversion, or the two types can be binary coercible.

  Typically, a transformation must have different source and target data types. However, if there is a conversion implementation function with more than one parameter, it is allowed to declare a conversion with the same source and target types. This is used to represent a length enforcement function of a specific type in the system catalog. The named function is used to force a value of this type to be the type modifier value given by the second parameter.

  If the source type and target type of a type conversion are different and more than one parameter is received, it indicates that only one step is required to convert one type to another and the length conversion is performed at the same time. If no such conversion is available, converting to a type that uses a type modifier involves two steps, one to convert between data types, and the other to apply a conversion specified by the modifier.

  Currently, domain type conversion does not take effect. Transformations are typically targeted to the domain-related data types to which they belong.

  NOTE

  Cast is executed depending on the permission of the user who invokes it. When invoking a cast created by others, check the execution content of the cast function to prevent the cast creator from performing unauthorized operations with the permission of the executor.

Examples

To create an assignment mapping from type bigint to type int4, use the int4(bigint) function:

```
CREATE CAST (bigint AS int4) WITH FUNCTION int4(bigint) AS ASSIGNMENT;
```

(The conversion has been predefined in the system.)

Compatibility

The CREATE CAST instruction complies with the SQL standard. Except that the SQL does not have extra parameters that can be forcibly converted to binary types or implement functions.

11.14.57 CREATE CLIENT MASTER KEY

Function

CREATE CLIENT MASTER KEY creates a CMK object that can be used to encrypt a CEK object.

Precautions

This syntax is specific to a fully-encrypted database.

When using **gsql** to connect to a database server, you need to use the **-C** parameter to enable the fully-encrypted database.

In the CMK object created using this syntax, only the method for reading keys from independent key management tools, services, or components is stored. The key itself is not stored.

Syntax

```
CREATE CLIENT MASTER KEY client_master_key_name WITH (KEY_STORE = key_store_name, KEY_PATH =
"key_path_value", ALGORITHM = algorithm_type)
```

Parameter Description

- **client_master_key_name**

  Specifies the key name. In the same namespace, the value of this parameter must be unique.

  Value range: a string. It must comply with the identifier naming convention.
- **KEY_STORE**

  Specifies the tool or service that independently manages keys. Currently, only the key management tool gs_ktool provided by GaussDB and the online key management service huawei_kms provided by Huawei Cloud are supported. Value range: **gs_ktool** and **huawei_kms**

  NOTICE

  Because only the client interacts with the **KEY_STORE**, the types supported by the **KEY_STORE** parameter in this syntax vary according to the client. When **gsql** is used to execute this syntax, **KEY_STORE** supports only **gs_ktool**. When JDBC is used to execute this syntax, **KEY_STORE** supports only **huawei_kms**.
- **KEY_PATH**

  Specifies a key in the key management tool or service. The **KEY_STORE** and **KEY_PATH** parameters can be used to uniquely identify a key entity. When **KEY_STORE** is set to **gs_ktool**, the value of **KEY_PATH** is **gs_ktool** or **KEY_ID**. When **KEY_STORE** is set to **huawei_kms**, the value is a 36-byte key ID.

NOTE

The CMK object created by this syntax stores the **KEY_STORE** and **KEY_PATH** information. When the key entity needs to be read, GaussDB can automatically read the specified key entity from the specified KEY_STORE based on the information stored in the CMK object. Therefore, in this syntax, the **KEY_PATH** parameter should point to an existing key entity.

- **ALGORITHM**

  Specifies the encryption algorithm used by the key entity. When **KEY_STORE** is set to **gs_ktool**, the value of **ALGORITHM** can be **AES_256_CBC** or **SM4**. When **KEY_STORE** is set to **huawei_kms**, the value of **ALGORITHM** is **AES_256**.

Example (Using gsql to Connect to the Database Server)

```
-- (1) Use the key management tool gs_ktool to create a key. The tool returns the ID of the newly generated key.
[cmd] gs_ktool -g

-- (2) Use a privileged account to create a common user named alice.
openGauss=# CREATE USER alice PASSWORD '********';
-- (3) Use the account of common user alice to connect to the encrypted database and execute the syntax.
gsql -p 57101 postgres -U alice -r -C
gsql((GaussDB Kernel VxxxRxxxCxx build f521c606) compiled at 2021-09-16 14:55:22 commit 2935 last mr 6385 release)
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

openGauss=>

-- Create a CMK object.
openGauss=> CREATE CLIENT MASTER KEY alice_cmk WITH ( KEY_STORE = gs_ktool , KEY_PATH = "gs_ktool/1" , ALGORITHM = AES_256_CBC);
```

Example (Using JDBC to Connect to the Database Server)

```
/*
 * (1) Log in to the Huawei Cloud official website (https://www.huaweicloud.com), choose Console > Service List > Data Encryption Workshop > Key Management Service, and create a key.
 *     KMS is a key management service provided by Huawei Cloud. You can also use APIs to manage keys. For details, see the following public document of Huawei Cloud:
 *    (https://support.huaweicloud.com/dew_faq/dew_01_0053.html)
 */

/*
 * (2) Establish a connection to the database server and execute the syntax. In the URL, set enable_ce to 1.
 *      Note: The code in this section is used as an example. Consider using the minimum code to implement the most basic functions.
 */
import java.sql.*;

public class CrtCmkTest {
    public static void main(String[] args) {
        String driver = "org.postgresql.Driver";
        try {
            Class.forName(driver);
        } catch (Exception e) {
            e.printStackTrace();
            return;
        }

        /* Information used to establish a connection to the database server */
        String dbUrl = "jdbc:postgresql://localhost:19900/postgres?enable_ce=1";
        String dbUser = "alice";
        String dbPassword = "********";
        /*
         * Used to access the identity authentication information of Huawei Cloud KMS and KMS project
information
         * Note: All parameters in this part can be found on the Console > My Credential page of the Huawei
Cloud official website.
         */
        String iamUser = "alice_for_kms";
        String iamPassword = "********";
        String kmsDomain = "hw00000000";
        String kmsProjectName = "cn-east-3";
        String kmsProjectId = "00000000000000000000000000000000";

        /* SQL statement for creating a CMK object */
        String sql = "CREATE CLIENT MASTER KEY alice_cmk WITH ( " +
         "KEY_STORE = huawei_kms, KEY_PATH = \"00000000-0000-0000-0000-000000000000\" ,
ALGORITHM = AES_256);";

        try {
            Connection conn = DriverManager.getConnection(dbUrl, dbUser, dbPassword);
            conn.setClientInfo("iamUser", iamUser);
            conn.setClientInfo("iamPassword", iamPassword);
            conn.setClientInfo("kmsDomain", kmsDomain);
            conn.setClientInfo("kmsProjectName", kmsProjectName);
            conn.setClientInfo("kmsProjectId", kmsProjectId );
            Statement stmt = conn.createStatement();
            System.out.println("results: " + stmt.executeUpdate(sql));
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

11.14.58 CREATE COLUMN ENCRYPTION KEY

Function

CREATE COLUMN ENCRYPTION KEY creates a CEK that can be used to encrypt a specified column in a table.

Precautions

This syntax is specific to a fully-encrypted database.

When using gsql to connect to a database server, you need to use the **-C** parameter to enable the fully-encrypted database.

The CEK object created using this syntax can be used for column-level encryption. When defining a column in a table, you can specify a CEK object to encrypt the column.

Syntax

```
CREATE COLUMN ENCRYPTION KEY column_encryption_key_name WITH VALUES(CLIENT_MASTER_KEY =
client_master_key_name, ALGORITHM = algorithm_type, ENCRYPTED_VALUE = encrypted_value);
```

Parameter Description

- **column_encryption_key_name**

  This parameter is used as the name of a key object. In the same namespace, the value of this parameter must be unique.

  Value range: a string. It must comply with the identifier naming convention.

- **CLIENT_MASTER_KEY**

  Specifies the CMK used to encrypt the CEK. The value is the CMK object name, which is created using the **CREATE CLIENT MASTER KEY** syntax.
- **ALGORITHM**

  Encryption algorithm to be used by the CEK. The value can be **AEAD_AES_256_CBC_HMAC_SHA256**, **AEAD_AES_128_CBC_HMAC_SHA256**, or **SM4_SM3**.
- **ENCRYPTED_VALUE (optional)**

  A key password specified by a user. The key password length ranges from 28 to 256 characters. The derived 28-character key meets the AES128 security requirements. If the user needs to use AES256, the key password length must be 39 characters. If the user does not specify the key password length, a 256-character key is automatically generated.

  NOTICE

  - SM algorithm constraints: SM2, SM3, and SM4 are Chinese national cryptography standards. To avoid legal risks, these algorithms must be used together. If you specify the SM4 algorithm to encrypt CEKs when creating a CMK, you must specify the SM3 and SM4 algorithms (SM4_SM3) to encrypt data when creating CEKs.
  - Constraints on the **ENCRYPTED_VALUE** field: If the CMK generated by Huawei KMS is used to encrypt the CEK and the **ENCRYPTED_VALUE** field is used to transfer the key in the **CREATE COLUMN ENCRYPTION KEY** syntax, the length of the input key must be an integer multiple of 16 bytes.

Examples

```
-- Create a CEK.
openGauss=> CREATE COLUMN ENCRYPTION KEY a_cek WITH VALUES (CLIENT_MASTER_KEY = a_cmk,
ALGORITHM  = AEAD_AES_256_CBC_HMAC_SHA256);
CREATE COLUMN ENCRYPTION KEY
openGauss=> CREATE COLUMN ENCRYPTION KEY another_cek WITH VALUES (CLIENT_MASTER_KEY =
a_cmk, ALGORITHM  = SM4_SM3);
CREATE COLUMN ENCRYPTION KEY
```

11.14.59 CREATE CONVERSION

Function

CREATE CONVERSION defines a new conversion between two character set encodings.

Precautions

- The **DEFAULT** parameter indicates that the conversion between the source encoding and the target encoding is executed by default between the client and the server. To support this usage, bidirectional conversion must be defined, that is, both conversion from A to B and conversion from B to A are supported.
- To perform conversion, you must have the EXECUTE permission on function and the CREATE permission on the target schema.

- SQL_ASCII cannot be used for either source encoding or target encoding because the server behavior is hardwired when SQL_ASCII "encoding" is involved.
- You can remove user-defined conversions using DROP CONVERSION.

Syntax

```
CREATE [ DEFAULT ] CONVERSION name
    FOR source_encoding TO dest_encoding FROM function_name
```

Parameter Description

- **DEFAULT**

  Specifies that the conversion is the default conversion from the source encoding to the target encoding. There should be only one default conversion for each encoding pair in a schema.
- **name**

  Specifies the name of the conversion, which can be restricted by the schema. If not restricted by a schema, the conversion is defined in the current schema. The conversion name must be unique in a schema.
- **source_encoding**

  Source encoding name.
- **dest_encoding**

  Target encoding name.
- **function_name**

  Function used for conversion. A function name can be restricted by a schema. If not, the function is found in the path.
  ```
  conv_proc(
      integer, -- Source encoding ID
      integer, -- Target encoding ID
      cstring, -- Source character string (C character string ending with a null value)
      internal,-- Target (filled with a null-terminated C character string)
      integer  -- Length of the source string
  ) RETURNS void;
  ```

Examples

```
-- Use myfunc to create an encoding conversion from UTF8 to LATIN1.
CREATE CONVERSION myconv FOR 'UTF8' TO 'LATIN1' FROM myfunc;
```

11.14.60 CREATE DATABASE

Function

CREATE DATABASE creates a database. By default, the new database will be created only by cloning the standard system database **template0**.

Precautions

- A user that has the CREATEDB permission or a system administrator can create a database.
- **CREATE DATABASE** cannot be executed inside a transaction block.

- During the database creation, an error message indicating that permission denied is displayed, possibly because the permission on the data directory in the file system is insufficient. If an error message, indicating no space left on device is displayed, the possible cause is that the disk space is used up.

Syntax

```
CREATE DATABASE database_name
    [ [ WITH ] { [ OWNER [=] user_name ] |
               [ TEMPLATE [=] template ] |
               [ ENCODING [=] encoding ] |
               [ LC_COLLATE [=] lc_collate ] |
               [ LC_CTYPE [=] lc_ctype ] |
               [ DBCOMPATIBILITY [=] compatibilty_type ] |
               [ TABLESPACE [=] tablespace_name ] |
               [ CONNECTION LIMIT [=] connlimit ]}[...] ];
```

Parameter Description

- **database_name**

  Specifies the database name.

  Value range: a string. It must comply with the identifier naming convention.

- **OWNER [=] user_name**

  Specifies the owner of the new database. If omitted, the default owner is the current user.

  Value range: an existing username

- **TEMPLATE [=] template**

  Specifies a template name. That is, the template from which the database is created. GaussDB creates a database by copying data from a template database. GaussDB has two default template databases **template0** and **template1** and a default user database **postgres**.

  Value range: **template0**

- **ENCODING [=] encoding**

  Specifies the character encoding used by the database. The value can be a string (for example, **SQL_ASCII**) or an integer.

  If this parameter is not specified, the encoding of the template database is used by default. By default, the codes of the template databases **template0** and **template1** are related to the operating system environment. The character encoding of **template1** cannot be changed. To change the encoding, use **template0** to create a database.

  Common values are **GBK**, **UTF8**, **Latin1**, and **GB18030**. The supported character sets are as follows:

Table 11-117 GaussDB character set

Name	Description	Language	Server-side Encoding	ICU Support	Number of Bytes/Characters	Alias
BIG5	Big Five	Traditional Chinese	No	No	1-2	WIN950, Windows950
EUC_CN	Extended UNIX Code-CN	Simplified Chinese	Yes	Yes	1-3	-
EUC_JP	Extended UNIX Code-JP	Japanese	Yes	Yes	1-3	-
EUC_JIS_2004	Extended UNIX Code-JP, JIS X 0213	Japanese	Yes	No	1-3	-
EUC_KR	Extended UNIX Code-KR	Korean	Yes	Yes	1-3	-
EUC_TW	Extended UNIX Code-Taiwan, China	Traditional Chinese	Yes	Yes	1-3	-
GB18030	National Standard	Chinese	Yes	No	1-4	-
GBK	Extended National Standard	Simplified Chinese	Yes	No	1-2	WIN936, Windows936
ISO_8859_5	ISO 8859-5, ECMA 113	Latin/Cyrillic	Yes	Yes	1	-
ISO_8859_6	ISO 8859-6, ECMA 114	Latin/Arabic	Yes	Yes	1	-

Name	Description	Language	Server-side Encoding	ICU Support	Number of Bytes/Characters	Alias
ISO_8859_7	ISO 8859-7, ECMA 118	Latin/Greek	Yes	Yes	1	-
ISO_8859_8	ISO 8859-8, ECMA 121	Latin/Hebrew	Yes	Yes	1	-
JOHAB	JOHAB	Korean	No	No	1-3	-
KOI8R	KOI8-R	Cyrillic (Russian)	Yes	Yes	1	KOI8
KOI8U	KOI8-U	Cyrillic (Ukrainian)	Yes	Yes	1	-
LATIN1	ISO 8859-1, ECMA 94	Western European	Yes	Yes	1	ISO88591
LATIN2	ISO 8859-2, ECMA 94	Central European	Yes	Yes	1	ISO88592
LATIN3	ISO 8859-3, ECMA 94	South European	Yes	Yes	1	ISO88593
LATIN4	ISO 8859-4, ECMA 94	North European	Yes	Yes	1	ISO88594
LATIN5	ISO 8859-9, ECMA 128	Turkish	Yes	Yes	1	ISO88599
LATIN6	ISO 8859-10, ECMA 144	Nordic	Yes	Yes	1	ISO885910

Name	Description	Language	Server-side Encoding	ICU Support	Number of Bytes/ Characters	Alias
LATIN7	ISO 8859-13	Baltic	Yes	Yes	1	ISO885913
LATIN8	ISO 8859-14	Celtic	Yes	Yes	1	ISO885914
LATIN9	ISO 8859-15	LATIN1 with Euro and accents	Yes	Yes	1	ISO885915
LATIN10	ISO 8859-16, ASRO SR 14111	Romanian	Yes	No	1	ISO885916
MULE_INTERNAL	Mule internal code	Multilingual Emacs	Yes	No	1-4	-
SJIS	Shift JIS	Japanese	No	No	1-2	Mskanji, ShiftJIS, WIN932, Windows932
SHIFT_JIS_2004	Shift JIS, JIS X 0213	Japanese	No	No	1-2	-
SQL_ASCII	Unspecified (see the text)	*Any*	Yes	No	1	-
UHC	Unified Hangul Code	Korean	No	No	1-2	WIN949, Windows949
UTF8	Unicode, 8-bit	***All***	Yes	Yes	1-4	Unicode
WIN866	Windows CP866	Cyrillic	Yes	Yes	1	ALT
WIN874	Windows CP874	Thai	Yes	No	1	-
WIN1250	Windows CP1250	Central European	Yes	Yes	1	-

Name	Description	Language	Server-side Encoding	ICU Support	Number of Bytes/Characters	Alias
WIN1251	Windows CP1251	Cyrillic	Yes	Yes	1	WIN
WIN1252	Windows CP1252	Western European	Yes	Yes	1	-
WIN1253	Windows CP1253	Greek	Yes	Yes	1	-
WIN1254	Windows CP1254	Turkish	Yes	Yes	1	-
WIN1255	Windows CP1255	Hebrew	Yes	Yes	1	-
WIN1256	Windows CP1256	Arabic	Yes	Yes	1	-
WIN1257	Windows CP1257	Baltic	Yes	Yes	1	-
WIN1258	Windows CP1258	Vietnamese	Yes	Yes	1	ABC, TCVN, TCVN5712, VSCII

CAUTION

Note that not all client APIs support the preceding character sets.

The SQL_ASCII setting performs quite differently from other settings. If the character set of the server is SQL_ASCII, the server interprets the byte values 0 to 127 according to the ASCII standard. The byte values 128 to 255 are regarded as the characters that cannot be parsed. If this parameter is set to SQL_ASCII, no code conversion occurs. Therefore, this setting is not basically used to declare the specified encoding used, because this declaration ignores the encoding. In most cases, if you use any non-ASCII data, it is unwise to use the SQL_ASCII setting because the database will not be able to help you convert or verify non-ASCII characters.

NOTICE

- The character set encoding of the new database must be compatible with the local settings (**LC_COLLATE** and **LC_CTYPE**).
- When the specified character encoding set is **GBK**, some uncommon Chinese characters cannot be directly used as object names. This is because the byte encoding overlaps with the ASCII characters @A-Z[\]^_`a-z{|} when the second byte of the GBK ranges from 0x40 to 0x7E. **@[\]^_'{|}** is an operator in the database. If it is directly used as an object name, a syntax error will be reported. For example, the GBK hexadecimal code is **0x8240**, and the second byte is **0x40**, which is the same as the ASCII character @. Therefore, the character cannot be used as an object name. If you do need to use this function, you can add double quotation marks ("") to avoid this problem when creating and accessing objects.
- If the client code is A and the server code is B, conversion between A and B must exist in the database. For example, when the encoding format on the server is GB18030 and that on the client is GBK, the error message "Conversion between GB18030 and GBK is not supported." will be displayed because the current database does not support the conversion between GB18030 and GBK. For details about conversion between the encoding formats supported by the database, see the **pg_conversion** system catalog.

- **LC_COLLATE [=] lc_collate**

  Specifies the character set used by the new database. For example, set this parameter by using **lc_collate = 'zh_CN.gbk'**.

  The use of this parameter affects the sort order of strings (for example, the order of using **ORDER BY** for execution and the order of using indexes on text columns). By default, the sorting order of the template database is used.

  Value range: character sets supported by the OS.

- **LC_CTYPE [=] lc_ctype**

  Specifies the character class used by the new database. For example, set this parameter by using **lc_ctype = 'zh_CN.gbk'**. The use of this parameter affects the classification of characters, such as uppercase letters, lowercase letters, and digits. By default, the character classification of the template database is used.

  Value range: character classes supported by the OS.

  NOTE

  The value ranges of **lc_collate** and **lc_ctype** depend on the character sets supported by the local environment. For example, in the Linux operating system, you can run the **locale -a** command to obtain the list of character sets supported by the operating system. When using the **lc_collate** and **lc_ctype** parameters, you can select the required character sets and character categories.

- **DBCOMPATIBILITY [=] compatibility_type**

  Specifies the compatible database type. The default compatible database is the **O** database.

  Value range: **A**, **B**, **C**, and **PG**, indicating the **O**, **MY**, **TD**, and **POSTGRES** databases, respectively.

NOTE

- For A compatibility, the database treats empty strings as **NULL** and replaces **DATE** with **TIMESTAMP(0) WITHOUT TIME ZONE**.
- When a character string is converted to an integer, if the input is invalid, the input will be converted to 0 due to B compatibility, and an error will be reported due to other compatibility issues.
- For PG compatibility, CHAR and VARCHAR are counted by character. For other compatibility types, they are counted by byte. For example, for the UTF-8 character set, CHAR(3) can store three Chinese characters in PG compatibility scenarios, but can store only one Chinese character in other compatibility scenarios.

- **TABLESPACE [=] tablespace_name**

  Specifies the tablespace of the database.

  Value range: an existing tablespace name

- **CONNECTION LIMIT [=] connlimit**

  Specifies the maximum number of concurrent connections that can be made to the new database.

  NOTICE

  - The system administrator is not restricted by this parameter.
  - connlimit is calculated separately for each primary database node. Number of connections of the database = connlimit x Number of normal primary database nodes.

  Value range: an integer greater than or equal to –1 The default value is **–1**, indicating that there is no limit.

The restrictions on character encoding are as follows:

- If the locale is set to **C** (or **POSIX**), all encoding types are allowed. For other locale settings, the character encoding must be the same as that of the locale.
- If the character encoding mode is SQL_ASCII and the modifier is an administrator, the character encoding mode can be different from the locale setting.
- The encoding and region settings must match the template database, except that **template0** is used as a template. This is because other databases may contain data that does not match the specified encoding, or may contain indexes whose sorting order is affected by **LC_COLLATE** and **LC_CTYPE**. Copying this data will invalidate the indexes in the new database. **template0** does not contain any data or indexes that may be affected.

Examples

```
-- Create users jim and tom.
openGauss=# CREATE USER jim PASSWORD 'xxxxxxxxx';
openGauss=# CREATE USER tom PASSWORD 'xxxxxxxxx';

-- Create database music using GBK (the local encoding type is also GBK).
openGauss=# CREATE DATABASE music ENCODING 'GBK' template = template0;

-- Create database music2 and specify user jim as its owner.
openGauss=# CREATE DATABASE music2 OWNER jim;
```

```
-- Create database music3 using template template0 and specify user jim as its owner.
openGauss=# CREATE DATABASE music3 OWNER jim TEMPLATE template0;

-- Set the maximum number of connections to database music to 10.
openGauss=# ALTER DATABASE music CONNECTION LIMIT= 10;

-- Rename database music to music4.
openGauss=# ALTER DATABASE music RENAME TO music4;

-- Change the owner of database music2 to user tom.
openGauss=# ALTER DATABASE music2 OWNER TO tom;

-- Set the tablespace of database music3 to PG_DEFAULT.
openGauss=# ALTER DATABASE music3 SET TABLESPACE PG_DEFAULT;

-- Disable the default index scan on database music3.
openGauss=# ALTER DATABASE music3 SET enable_indexscan TO off;

-- Reset the enable_indexscan parameter.
openGauss=# ALTER DATABASE music3 RESET enable_indexscan;

-- Delete the database.
openGauss=# DROP DATABASE music2;
openGauss=# DROP DATABASE music3;
openGauss=# DROP DATABASE music4;

-- Delete the jim and tom users.
openGauss=# DROP USER jim;
openGauss=# DROP USER tom;

-- Create a database compatible with the TD format.
openGauss=# CREATE DATABASE td_compatible_db DBCOMPATIBILITY 'C';

-- Create a database compatible with the A format.
openGauss=# CREATE DATABASE ora_compatible_db DBCOMPATIBILITY 'A';

-- Delete the databases that are compatible with the TD and A formats.
openGauss=# DROP DATABASE td_compatible_db;
openGauss=# DROP DATABASE ora_compatible_db;
```

Helpful Links

ALTER DATABASE and DROP DATABASE

Suggestions

- **create database**

  Database cannot be created in a transaction.

- **ENCODING LC_COLLATE LC_CTYPE**

  If the setting of **Encoding** of the new database does not match that of the template database (SQL_ASCII) (**'GBK'**, **'UTF8'**, **'LATIN1'**, or **'GB18030'**), **template [=] template0** must be specified.

11.14.61 CREATE DATA SOURCE

Function

CREATE DATA SOURCE creates an external data source, which defines the information about the database that GaussDB will connect to.

Precautions

- The data source name must be unique in the database and comply with the identifier naming rules. Its length cannot exceed 63 bytes. Otherwise, it will be truncated.
- Only the system administrator or initial user has the permission to create data sources. The user who creates the object is the default owner of the object.
- If the **password** option is displayed, ensure that the **datasource.key.cipher** and **datasource.key.rand** files exist in the *$GAUSSHOME***/bin** directory of each node in the database. If the two files do not exist, use the **gs_guc** tool to generate them and use the **gs_ssh** tool to release them to the *$GAUSSHOME***/bin** directory on each node.

Syntax

```
CREATE DATA SOURCE src_name
    [TYPE 'type_str']
    [VERSION {'version_str' | NULL}]
    [OPTIONS (optname 'optvalue' [, ...])];
```

Parameter Description

- **src_name**

  Specifies the name of the new data source, which must be unique in the database.

  Value range: a string compliant with the identifier naming convention
- **TYPE**

  Specifies the type of the data source. This parameter can be left empty, and its default value will be used.

  Value range: an empty string or a non-empty string
- **VERSION**

  Specifies the version number of the new data source. This parameter can be left empty or set to null.

  Value range: an empty string, a non-empty string, or null
- **OPTIONS**

  Specifies the options of the data source. This parameter can be left empty or specified using the following keywords:

  – optname

    Specifies the option name.

    Value range: **dsn**, **username**, **password**, and **encoding**. The value is case-insensitive.

    ▪ **dsn** corresponds to the DSN in the ODBC configuration file.

    ▪ **username**/**password** indicates the username and password for connecting to the destination database.

      The user name and password entered by the user are encrypted in the GaussDB background to ensure security. The key file required for encryption must be generated using the **gs_guc** tool and released to the *$GAUSSHOME***/bin** directory of each node in the database using the **gs_ssh** tool. The user name and password cannot contain the prefix "encryptOpt". Otherwise, they are considered as encrypted ciphertext.

- **encoding** indicates the character string encoding mode used for interaction with the destination database (including the sent SQL statements and returned data of the character type). Its validity is not checked during object creation. Whether data can be encoded and decoded depends on whether the encoding you specified can be used in the database.

– optvalue

Specifies the option value.

Value range: an empty string or a non-empty string

Examples

```
-- Create an empty data source that does not contain any information.
openGauss=# CREATE DATA SOURCE ds_test1;

-- Create a data source with TYPE information and VERSION being null.
openGauss=# CREATE DATA SOURCE ds_test2 TYPE 'MPPDB' VERSION NULL;

-- Create a data source that contains only OPTIONS.
openGauss=#  CREATE DATA SOURCE ds_test3 OPTIONS (dsn 'GaussDB', encoding 'utf8');

-- Create a data source that contains TYPE, VERSION, and OPTIONS.
openGauss=# CREATE DATA SOURCE ds_test4 TYPE 'unknown' VERSION '11.2.3' OPTIONS (dsn 'GaussDB', 
username 'userid', password 'pwd@123456', encoding '');

-- Delete the data source.
openGauss=# DROP DATA SOURCE ds_test1;
openGauss=# DROP DATA SOURCE ds_test2;
openGauss=# DROP DATA SOURCE ds_test3;
openGauss=# DROP DATA SOURCE ds_test4;
```

Helpful Links

ALTER DATA SOURCE and **DROP DATA SOURCE**

11.14.62 CREATE DIRECTORY

Function

CREATE DIRECTORY creates a directory. The directory defines an alias for a path in the server file system and is used to store data files used by users.

Precautions

- When **enable_access_server_directory** is set to **off**, only the initial user is allowed to create directory objects. When **enable_access_server_directory** is set to **on**, the user with the SYSADMIN permission and the user who inherits the **gs_role_directory_create** permission of the built-in role can create directory objects.
- By default, the user who creates a directory has the read and write permissions on the directory.
- The default owner of a directory is the user who creates the directory.
- A directory cannot be created for the following paths:

  - The path contains special characters.
  - The path is a relative path.
  - The path is a symbolic link.
- The following validity check is performed during directory creation:
  - Check whether the path exists in the OS. If it does not exist, a message is displayed, indicating the potential risks.
  - Check whether the database initial user **omm** has the R/W/X permissions for the OS path. If the user does not have all the permissions, a message is displayed, indicating the potential risks.
- In the database, ensure that the path is the same on all the nodes. Otherwise, the path may fail to be found on some nodes when the directory is used.

Syntax

```
CREATE [OR REPLACE] DIRECTORY directory_name
AS 'path_name';
```

Parameter Description

- **directory_name**

  Specifies the name of a directory.

  Value range: a string. It must comply with the identifier naming convention.
- **path_name**

  Specifies the OS path for which a directory is to be created.

  Value range: a valid OS path

Examples

```
-- Create a directory.
openGauss=# CREATE OR REPLACE DIRECTORY  dir  as '/tmp/';
```

Helpful Links

ALTER DIRECTORY and **DROP DIRECTORY**

11.14.63 CREATE EXTENSION

Function

CREATE EXTENSION installs an extension.

Precautions

- Before running the **CREATE EXTENSION** command to load an extension into the database, you must install the support file for the extension first.
- The **CREATE EXTENSION** command installs a new extension to a database. Ensure that no extension with the same name has been installed.
- Installing an extension means executing an extended script file that creates a SQL entity, such as a function, data type, operator, and index-supported method.

- Installing an extension requires the same permissions as creating its component objects. For most extensions, this means that the superuser or database owner's permissions are required. For subsequent permission checks and entities created by the extension script, the role that runs the **CREATE EXTENSION** command becomes the owner of the extension.
- During the execution of **CREATE EXTENSION**, if the database contains database objects with the same name as those in the EXTENSION, such as packages, synonyms, operators, directories, functions, stored procedures, views, and tables, the execution will fail.

Syntax

```
CREATE EXTENSION [ IF NOT EXISTS ] extension_name
[ WITH ] [ SCHEMA schema_name ]
[ VERSION version ]
[ FROM old_version ]
```

Parameter Description

- **IF NOT EXISTS**

  If an extension with the same name exists in the system, no error is reported. However, a message is displayed. Note that this parameter does not ensure that the existing extensions of the system are the same as those created by the script.

- **extension_name**

  Specifies the name of the extension to be installed. The database creates the extension by using the information in the **SHAREDIR/extension/extension_name.control** file.

- **schema_name**

  The extension instance is installed in this schema, and the extended content can be reinstalled. The specified schema must exist. If it is not specified, the extended control file does not specify a schema either. In this case, the default schema is used.

  CAUTION

  Extensions are not considered to be in any schema (no restriction is posed on the name of extensions within the scope of a database), but an extension instance belongs to a schema.

- **version**

  Version of the extension to be installed, which can be written as an identifier or a string. The default version is specified in the extended control file.

- **old_version**

  If you want to upgrade the content that is not contained in the **old style** module, you must specify **FROM old_version**. This option makes **CREATE EXTENSION** run an installation script to install new content into the extension instead of creating an entity. Note that **SCHEMA** specifies the schema that includes these existing entities.

Examples

Install the **hstore** extension in the current database.

```
CREATE EXTENSION hstore;
```

11.14.64 CREATE FOREIGN TABLE

Function

CREATE FOREIGN TABLE creates a foreign table.

Precautions

System columns (such as **tableoid** and **ctid**) cannot be used in foreign tables. Foreign tables in the Private or Shared schema require the initial user permission or the O&M administrator permission in O&M mode (operation_mode).

When multi-layer quotation marks are used for sensitive columns (such as **password** and **secret_access_key**) in OPTIONS, the semantics is different from that in the scenario where quotation marks are not used. Therefore, sensitive columns are not identified for anonymization.

Syntax

```
CREATE FOREIGN TABLE [ IF NOT EXISTS ] table_name ( [
    column_name type_name [ OPTIONS ( option 'value' [, ... ] ) ] [ COLLATE collation ] [ column_constraint [ ... ] ]
    [, ... ]
] )
  SERVER server_name
[ OPTIONS ( option 'value' [, ... ] ) ]

The column_constraint can be:
[ CONSTRAINT constraint_name ]
{ NOT NULL |
  NULL |
  DEFAULT default_expr }
```

Parameter Description

- **IF NOT EXISTS**

  Sends a notice, but does not throw an error, if a table with the same name exists.
- **table_name**

  Specifies the name of a foreign table.

  Value range: a string. It must comply with the identifier naming convention.
- **column_name**

  Specifies the name of a column in the foreign table.

  Value range: a string. It must comply with the identifier naming convention.
- **type_name**

  Specifies the data type of the column.
- **SERVER server_name**

  Specifies the server name of the foreign table.

- **OPTIONS (option 'value' [, ...])**

  Options are related to the new foreign table or the columns in the foreign table. The allowed option names and values are specified by each foreign data wrapper, and are also verified by the verification function of the foreign data wrapper. The option name must be unique (although table options and table column options can share the same name).

  - Options supported by **file_fdw** are as follows:
    - filename

      File to be read. This parameter is mandatory and must be an absolute path.
    - format

      File format of the remote server, which is the same as the **FORMAT** option in the **COPY** statement. The value can be **text**, **csv**, **binary**, or **fixed**.
    - header

      Specifies whether a specified file has a header, which is the same as the **HEADER** option of the **COPY** statement.
      - delimiter

        File delimiter, which is the same as the **DELIMITER** option of the **COPY** statement.
      - quote

        Quote character of a file, which is the same as the **QUOTE** option of the **COPY** statement.
      - escape

        Escape character of a file, which is the same as the **ESCAPE** option of the **COPY** statement.
      - null

        Null string of a file, which is the same as the **NULL** option of the **COPY** statement.
      - encoding

        Encoding of a file, which is the same as the **ENCODING** option of the **COPY** statement.
      - force_not_null

        This is a Boolean option. If it is true, the value of the declared field cannot be an empty string. This option is the same as the **FORCE_NOT_NULL** option of the **COPY** statement.

Helpful Links

ALTER FOREIGN TABLE and **DROP FOREIGN TABLE**.

11.14.65 CREATE FUNCTION

Function

CREATE FUNCTION creates a function.

Precautions

- If the parameters or return values of a function have precision, the precision is not checked.
- When creating a function, you are advised to explicitly specify the schemas of tables in the function definition. Otherwise, the function may fail to be executed.
- **current_schema** and **search_path** specified by **SET** during function creation are invalid. **search_path** and **current_schema** before and after function execution should be the same.
- If a function has output parameters, the **SELECT** statement uses the default values of the output parameters when calling the function. When the **CALL** statement calls the function, it requires that the output parameters must be specified. When the **CALL** statement calls an overloaded **PACKAGE** function, it can use the default values of the output parameters. For details, see examples in **CALL**.
- Only the functions compatible with PostgreSQL or those with the **PACKAGE** attribute can be overloaded. After **REPLACE** is specified, a new function is created instead of replacing a function if the number of parameters, parameter type, or return value is different.
- You can use the **SELECT** statement to specify different parameters using identical functions, but cannot use the **CALL** statement to call identical functions without the **PACKAGE** attribute.
- When you create a function, you cannot insert other agg functions out of the avg function or other functions.
- By default, the permissions to execute new functions are granted to PUBLIC users. For details, see **GRANT**. By default, a user inherits the permissions of the PUBLIC role. Therefore, the user has the permission to execute a function and view the definition of the function. In addition, to execute the function, the user must have the USAGE permission on the schema to which the function locates. You can revoke the default execution permissions from the PUBLIC role when creating a function and grant the function execution permission to users as needed. To avoid the time window during which new functions can be accessed by all users, create functions and set function execution permissions in a transaction. After the database object isolation attribute is enabled, common users can view only the definitions of functions that they have the permission to execute. For details about how to enable the attribute, see *Security Hardening Guide*.
- When functions without parameters are called inside another function, you can omit brackets and call functions using their names directly.
- When functions with output parameters are called inside another function which is an assignment expression, you can omit the output parameters of the called functions.

- Oracle-compatible functions support viewing, exporting, and importing parameter comments.
- Oracle-compatible functions support viewing, exporting, and importing comments between IS/AS and plsql_body.
- Users granted with the CREATE ANY FUNCTION permission can create or replace functions in the user schemas.
- The default permission on a function is SECURITY INVOKER. To change the permission to SECURITY DEFINER, set the GUC parameter **behavior_compat_options** to **'plsql_security_definer'**.
- For PL/pgSQL functions, after **behavior_compat_options** is set to **'proc_outparam_override'**, the behavior of **out/inout** changes. In the functions, **return** and **out/inout** can be returned at the same time. Before the parameter is enabled, only **return** is returned. For details, see **Examples**.
- For PL/pgSQL functions, after **behavior_compat_options** is set to **'proc_outparam_override'**, the restrictions are as follows:
  a. If a function with the **out/inout** parameter already exists in the same schema or package, you cannot create another function with the same name with the **out/inout** parameter.
  b. The **out** parameter must be added no matter whether the **SELECT** or **CALL** statement is used to call a stored procedure.
  c. In some scenarios, functions cannot be used in expressions (compared with those before the parameter is enabled), for example, left assignment in a stored procedure and **call function**. For details, see **Examples**.
  d. Functions without **return** cannot be called. **perform function** can be used to call functions.
  e. When a function is called in a stored procedure, **out/inout** cannot be set to a constant. For details, see **Examples**.

Syntax

- Syntax (compatible with PostgreSQL) for creating a customized function:

```
CREATE [ OR REPLACE  ] FUNCTION function_name
    [ ( [  { argname [ argmode  ] argtype [  { DEFAULT  | :=  | =  } expression  ]}  [, ...]  ] ) ]
    [ RETURNS rettype [ DETERMINISTIC  ]  | RETURNS TABLE (  { column_name column_type  }
[, ...] )]
    LANGUAGE lang_name
    [
       {IMMUTABLE  | STABLE  | VOLATILE }
        | {SHIPPABLE | NOT SHIPPABLE}
        | WINDOW
        | [ NOT  ] LEAKPROOF
        | {CALLED ON NULL INPUT  | RETURNS NULL ON NULL INPUT | STRICT }
        | {[ EXTERNAL  ] SECURITY INVOKER | [ EXTERNAL  ] SECURITY DEFINER | AUTHID DEFINER  |
AUTHID CURRENT_USER}
        | {fenced | not fenced}
        | {PACKAGE}
        | COST execution_cost
        | ROWS result_rows
        | SET configuration_parameter { {TO | =} value | FROM CURRENT }}
     ][...]
    {
        AS 'definition'
    }
```

- A-style database syntax of creating a customized function:

```
CREATE [ OR REPLACE  ] FUNCTION function_name
    ( [  { argname [ argmode  ] argtype [  { DEFAULT | := | =  } expression  ] }  [, ...]  ] )
```

```
  RETURN rettype [ DETERMINISTIC  ]
  [
      {IMMUTABLE  | STABLE  | VOLATILE }
      | {SHIPPABLE | NOT SHIPPABLE}
      | {PACKAGE}
      | {FENCED | NOT FENCED}
      | [ NOT  ] LEAKPROOF
      | {CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT }
      | {[ EXTERNAL  ] SECURITY INVOKER  | [ EXTERNAL  ] SECURITY DEFINER |
AUTHID DEFINER | AUTHID CURRENT_USER
}
      | COST execution_cost
      | ROWS result_rows
      | SET configuration_parameter { {TO | =} value  | FROM CURRENT
      | LANGUAGE lang_name
  ][...]

  {
    IS  | AS
} plsql_body
/
```

Parameter Description

- **function_name**

  Specifies the name of the function to create (optionally schema-qualified).

  Value range: a string. It must comply with the identifier naming convention, and can contain a maximum of 63 characters. If the value contains more than 63 characters, the database truncates it and retains the first 63 characters as the function name.
- **argname**

  Specifies the parameter name of the function.

  Value range: a string. It must comply with the identifier naming convention, and can contain a maximum of 63 characters. If the value contains more than 63 characters, the database truncates it and retains the first 63 characters as the function parameter name.
- **argmode**

  Specifies the parameter mode of the function.

  Value range: **IN**, **OUT**, **INOUT**, and **VARIADIC**. The default value is **IN**. The parameters of **OUT** and **INOUT** cannot be used in the function definition of **RETURNS TABLE**.

  NOTE

  VARIADIC specifies parameters of the array type.
- **argtype**

  Specifies the data type of a function parameter. **%TYPE** or **%ROWTYPE** can be used to indirectly reference a variable or table type. For details, see **Variable Definition Statements**.
- **expression**

  Specifies the default expression of a parameter.
- **rettype**

  Specifies the return data type.

  When there is **OUT** or **INOUT** parameter, the **RETURNS** clause can be omitted. If the clause exists, the result type of the clause must be the same as that of the output parameter. If there are multiple output parameters, the result type of the clause is **RECORD**. Otherwise, the result type of the clause is the same as that of a single output parameter.

The **SETOF** modifier indicates that the function will return a set of items, rather than a single item.

Same as **argtype**, **%TYPE** or **%ROWTYPE** can also be used to indirectly reference types.

NOTE

In **FUNCTION argtype** and **rettype** outside **PACKAGE**, **%TYPE** cannot reference the type of the *PACKAGE* variable.

- **column_name**

  Specifies the column name.
- **column_type**

  Specifies the column type.
- **definition**

  Specifies a string constant defining a function. Its meaning depends on the language. It can be an internal function name, a path pointing to a target file, a SQL query, or text in a procedural language.
- **DETERMINISTIC**

  Specifies an interface compatible with the SQL syntax. You are not advised to use it.
- **LANGUAGE lang_name**

  Specifies the name of the language that is used to implement the function. It can be **SQL**, **internal**, or the name of a customized process language. To ensure downward compatibility, the name can use single quotation marks. Contents in single quotation marks must be capitalized.

  Due to compatibility issues, no matter which language is specified when an A-style database is created, the language used is plpgsql.
- **WINDOW**

  Indicates that this function is a window function. The **WINDOW** attribute cannot be changed when replacing an existing function definition.

NOTICE

For a customized window function, the value of **LANGUAGE** can only be **internal**, and the referenced internal function must be a window function.

- **IMMUTABLE**

  Specifies that the function always returns the same result if the parameter values are the same.
- **STABLE**

  Specifies that the function cannot modify the database, and that within a single table scan it will consistently return the same result for the same parameter value, but its result varies by SQL statements.
- **VOLATILE**

Specifies that the function value can change in a single table scan and no optimization is performed.

- **SHIPPABLE|NOT SHIPPABLE**

  Specifies whether the function can be pushed down for execution. This port is reserved and is not recommended.

- **FENCED|NOT FENCED**

  Specifies whether the user-defined C function is executed in fenced or not-fenced mode. This port is reserved and is not recommended.

- **PACKAGE**

  Specifies whether the function can be overloaded. PostgreSQL-style functions can be overloaded, and this parameter is designed for functions of other styles.

  - All PACKAGE and non-PACKAGE functions cannot be overloaded or replaced.
  - PACKAGE functions do not support parameters of the VARIADIC type.
  - The **PACKAGE** attribute of functions cannot be modified.

- **LEAKPROOF**

  Specifies that the function has no side effects. **LEAKPROOF** can be set only by the system administrator.

- **CALLED ON NULL INPUT**

  Declares that some parameters of the function can be invoked in normal mode if the parameter values are null. This parameter can be omitted.

- **RETURNS NULL ON NULL INPUT**

  STRICT

  Specifies that the function always returns null whenever any of its parameters is null. If this parameter is specified, the function is not executed when there are null parameters; instead a null result is returned automatically.

  RETURNS NULL ON NULL INPUT and **STRICT** have the same functions.

- **EXTERNAL**

  The keyword **EXTERNAL** is allowed for SQL conformance, but it is optional since, unlike in SQL, this feature applies to all functions not only external ones.

- **SECURITY INVOKER**

  AUTHID CURRENT_USER

  Specifies that the function will be executed with the permissions of the user who invokes it. This parameter can be omitted.

  SECURITY INVOKER and **AUTHID CURRENT_USER** have the same functions.

- **SECURITY DEFINER**

  AUTHID DEFINER

  Specifies that the function will be executed with the permissions of the user who created it.

  AUTHID DEFINER and **SECURITY DEFINER** have the same functions.

- **COST execution_cost**

  Estimates the execution cost of a function.

The unit of **execution_cost** is **cpu_operator_cost**.

Value range: a positive integer

- **ROWS result_rows**

  Estimates the number of rows returned by the function. This is only allowed when the function is declared to return a set.

  Value range: a positive number. The default value is **1000**.

- **configuration_parameter**
  - **value**

    Sets a specified database session parameter to a specified value. If the value is **DEFAULT** or **RESET**, the default setting is used in the new session. **OFF** closes the setting.

    Value range: a string

    - DEFAULT
    - OFF
    - RESET

    Specifies the default value.
  - **from current**

    Uses the value of **configuration_parameter** of the current session.

- **plsql_body**

  Specifies the PL/SQL stored procedure body.

NOTICE

When a user is created in the function body, the plaintext password is recorded in the log. You are not advised to do it.

Examples

```
-- Define a function as SQL query.
openGauss=# CREATE FUNCTION func_add_sql(integer, integer) RETURNS integer
    AS 'select $1 + $2;'
    LANGUAGE SQL
    IMMUTABLE
    RETURNS NULL ON NULL INPUT;

-- Add an integer by parameter name using PL/pgSQL.
openGauss=# CREATE OR REPLACE FUNCTION func_increment_plsql(i integer) RETURNS integer AS $$
        BEGIN
                RETURN i + 1;
        END;
$$ LANGUAGE plpgsql;

-- Return the RECORD type.
openGauss=# CREATE OR REPLACE FUNCTION func_increment_sql(i int, out result_1 bigint, out result_2
bigint)
returns SETOF RECORD
as $$
begin
    result_1 = i + 1;
    result_2 = i * 10;
```

```
return next;
end;
$$language plpgsql;

-- Return a record containing multiple output parameters.
openGauss=# CREATE FUNCTION func_dup_sql(in int, out f1 int, out f2 text)
    AS $$ SELECT $1, CAST($1 AS text) || ' is text' $$
    LANGUAGE SQL;

openGauss=# SELECT * FROM func_dup_sql(42);

-- Compute the sum of two integers and returning the result (if the input is null, the returned result is null):
openGauss=# CREATE FUNCTION func_add_sql2(num1 integer, num2 integer) RETURN integer
AS
BEGIN PAC
RETURN num1 + num2;
END;
/
-- Alter the execution rule of function func_add_sql2 to IMMUTABLE (that is, the same result is returned if the parameter remains unchanged).
openGauss=# ALTER FUNCTION func_add_sql2(INTEGER, INTEGER) IMMUTABLE;

-- Rename the func_add_sql2 function as add_two_number:
openGauss=# ALTER FUNCTION func_add_sql2(INTEGER, INTEGER) RENAME TO add_two_number;

-- Change the owner of function add_two_number to omm.
openGauss=# ALTER FUNCTION omm(INTEGER, INTEGER) OWNER TO omm;

-- Delete the function.
openGauss=# DROP FUNCTION add_two_number;
openGauss=# DROP FUNCTION func_increment_sql;
openGauss=# DROP FUNCTION func_dup_sql;
openGauss=# DROP FUNCTION func_increment_plsql;
openGauss=# DROP FUNCTION func_add_sql;

-- Set parameters.
openGauss=# set behavior_compat_options='proc_outparam_override';
-- Create functions.
openGauss=# CREATE or replace FUNCTION func1(in a integer, out b integer)
RETURNS int
AS $$
DECLARE
    c int;
    BEGIN
        c := 1;
        b := a + c;
        return c;
    END; $$
LANGUAGE 'plpgsql' NOT FENCED;
-- Return return and output parameters at the same time.
openGauss=# declare
    result integer;
    a integer := 2;
    b integer := NULL;
begin
    result := func1(a => a, b => b);
    raise info 'b is: %', b;
    raise info 'result is: %', result;
end;
/
INFO:  b is: 3
INFO:  result is: 1
ANONYMOUS BLOCK EXECUTE
-- Left assignment expressions are not supported.
openGauss=# declare
    result integer;
    a integer := 2;
    b integer := NULL;
begin
```

```
    result := func1(a => a, b => b) + 1;
    raise info 'b is: %', b;
    raise info 'result is: %', result;
end;
/
ERROR:  when invoking function func1, maybe input something superfluous.
CONTEXT:  compilation of PL/pgSQL function "inline_code_block" near line 3
-- out/inout in a stored procedure cannot be set to a constant.
openGauss=# declare
    result integer;
    a integer := 2;
    b integer := NULL;
begin
    result := func1(a => a, b => 10);
    raise info 'b is: %', b;
    raise info 'result is: %', result;
end;
/
ERROR:  when invoking function func1, no destination for argments "b"
CONTEXT:  compilation of PL/pgSQL function "inline_code_block" near line 3
```

Helpful Links

ALTER FUNCTION and **DROP FUNCTION**.

11.14.66 CREATE GROUP

Function

CREATE GROUP creates a user group.

Precautions

CREATE GROUP is an alias for **CREATE ROLE**, and it is not a standard SQL syntax and not recommended. Users can use **CREATE ROLE** directly.

Syntax

```
CREATE GROUP group_name [ [ WITH ] option [ ... ] ] [ ENCRYPTED | UNENCRYPTED ] { PASSWORD |
IDENTIFIED BY } { 'password' [ EXPIRED ] | DISABLE };
```

The syntax of the **option** clause is as follows:

```
{SYSADMIN | NOSYSADMIN}
    | {MONADMIN | NOMONADMIN}
    | {OPRADMIN | NOOPRADMIN}
    | {POLADMIN | NOPOLADMIN}
    | {AUDITADMIN | NOAUDITADMIN}
    | {CREATEDB | NOCREATEDB}
    | {USEFT | NOUSEFT}
    | {CREATEROLE | NOCREATEROLE}
    | {INHERIT | NOINHERIT}
    | {LOGIN | NOLOGIN}
    | {REPLICATION | NOREPLICATION}
    | {INDEPENDENT | NOINDEPENDENT}
    | {VCADMIN | NOVCADMIN}
    | {PERSISTENCE | NOPERSISTENCE}
    | CONNECTION LIMIT connlimit
    | VALID BEGIN 'timestamp'
    | VALID UNTIL 'timestamp'
    | RESOURCE POOL 'respool'
    | PERM SPACE 'spacelimit'
    | TEMP SPACE 'tmpspacelimit'
    | SPILL SPACE 'spillspacelimit'
    | IN ROLE role_name [, ...]
    | IN GROUP role_name [, ...]
    | ROLE role_name [, ...]
    | ADMIN rol e_name [, ...]
    | USER role_name [, ...]
    | SYSID uid
    | DEFAULT TABLESPACE tablespace_name
    | PROFILE DEFAULT
    | PROFILE profile_name
    | PGUSER
```

Parameter Description

See **Parameter Description** in **CREATE ROLE**.

Helpful Links

ALTER GROUP, **DROP GROUP**, and **CREATE ROLE**

11.14.67 CREATE INCREMENTAL MATERIALIZED VIEW

Function

CREATE INCREMENTAL MATERIALIZED VIEW creates a fast-refresh materialized view, and you can refresh the data of the materialized view by using **REFRESH MATERIALIZED VIEW** (full refresh) and **REFRESH INCREMENTAL MATERIALIZED VIEW** (incremental refresh).

CREATE INCREMENTAL MATERIALIZED VIEW is similar to **CREATE TABLE AS**, but it remembers the query used to initialize the view, so it can refresh data later. A materialized view has many attributes that are the same as those of a table, but does not support temporary materialized views.

Precautions

- Fast-refresh materialized views cannot be created on temporary tables or global temporary tables.
- Fast-refresh materialized views support only simple filter queries and UNION ALL queries of base tables.
- Distribution columns cannot be specified when an incremental MV is created.
- After a fast-refresh materialized view is created, most DDL operations in the base table are no longer supported.
- IUD operations cannot be performed on fast-refresh materialized views.
- After a fast-refresh materialized view is created, you need to run the **REFRESH** command to synchronize the materialized view with the base table when the base table data changes.

Syntax

```
CREATE INCREMENTAL MATERIALIZED VIEW mv_name
    [ (column_name [, ...] ) ]
    [ TABLESPACE tablespace_name ]
    AS query;
```

Parameter Description

- **mv_name**

  Name (optionally schema-qualified) of the materialized view to be created.

  Value range: a string. It must comply with the naming convention.

- **column_name**

  Column name in the new materialized view. The materialized view supports specified columns. The number of specified columns must be the same as the number of columns in the result of the subsequent query statement. If no column name is provided, the column name is obtained from the output column name of the query.

  Value range: a string. It must comply with the naming convention.

- **TABLESPACE tablespace_name**

  Tablespace to which the new materialized view belongs. If not specified, the default tablespace is used.

- **AS query**

  SELECT or **TABLE** command. This query will be run in a security-constrained operation.

Examples

```
-- Create an ordinary table.
openGauss=# CREATE TABLE my_table (c1 int, c2 int);
-- Create a fast-refresh materialized view.
openGauss=# CREATE INCREMENTAL MATERIALIZED VIEW my_imv AS SELECT * FROM my_table;
-- Write data to the base table.
openGauss=# INSERT INTO my_table VALUES(1,1),(2,2);
-- Incrementally refresh the fast-refresh materialized view my_imv.
openGauss=# REFRESH INCREMENTAL MATERIALIZED VIEW my_imv;
```

Helpful Links

ALTER MATERIALIZED VIEW, **CREATE MATERIALIZED VIEW**, **CREATE TABLE**, **DROP MATERIALIZED VIEW**, **REFRESH INCREMENTAL MATERIALIZED VIEW**, and **REFRESH MATERIALIZED VIEW**

11.14.68 CREATE INDEX

Function

CREATE INDEX creates an index in a specified table.

Indexes are primarily used to enhance database performance (though inappropriate use can result in database performance deterioration). You are advised to create indexes on:

- Columns that are often queried
- Join conditions. For a query on joined columns, you are advised to create a composite index on the columns. For example, for **select * from t1 join t2 on t1.a=t2.a and t1.b=t2.b**, you can create a composite index on columns **a** and **b** in table **t1**.
- Columns having filter criteria (especially scope criteria) of a **where** clause

- Columns that appear after **order by**, **group by**, and **distinct**

Partitioned tables do not support partial index creation when indexes contain the GLOBAL or LOCAL keyword or the created index is a GLOBAL index.

Precautions

- Indexes consume storage and computing resources. Creating too many indexes has negative impact on database performance (especially the performance of data import. Therefore, you are advised to import the data before creating indexes). Therefore, create indexes only when they are necessary.
- All functions and operators used in an index definition must be immutable, that is, their results must depend only on their parameters and never on any outside influence (such as the contents of another table or the current time). This restriction ensures that the behavior of the index is well-defined. To use a customized function in an index expression or **WHERE** clause, remember to mark the function **immutable** when you create it.
- Partitioned table indexes are classified into local indexes and global indexes. A local index binds to a specific partition, and a global index corresponds to the entire partitioned table.
- If the two indexes are used, you cannot create expression and partial indexes. If the PSORT index is used, you cannot create unique indexes. If the B-tree index is used, you can create unique indexes.
- Column-store tables support GIN indexes, rather than partial indexes and unique indexes. If GIN indexes are used, you can create expression indexes. However, an expression in this situation cannot contain empty splitters, empty columns, or multiple columns.
- A user granted with the CREATE ANY INDEX permission can create indexes in both the public and user schemas.
- If a user-defined function is called in the expression index, the expression index function is executed based on the permission of the function creator.

Syntax

- Create an index on a table.
```
CREATE [ UNIQUE ] INDEX [ CONCURRENTLY ] [ [schema_name.]index_name ] ON table_name
[ USING method ]
    ({ { column_name | ( expression ) } [ COLLATE collation ] [ opclass ] [ ASC | DESC ] [ NULLS
{ FIRST | LAST } ] }[, ...] )
    [ INCLUDE ( column_name [, ...] )]
    [ WITH ( {storage_parameter = value} [, ... ] ) ]
    [ TABLESPACE tablespace_name ]
    [ WHERE predicate ];
```
- Create an index on a partitioned table.
```
CREATE [ UNIQUE ] INDEX [ [schema_name.]index_name ] ON table_name [ USING method ]
    ( {{ column_name | ( expression ) } [ COLLATE collation ] [ opclass ] [ ASC | DESC ] [ NULLS
LAST ] }[, ...] )
    [ LOCAL [ ( { PARTITION index_partition_name | SUBPARTITION index_subpartition_name
[ TABLESPACE index_partition_tablespace ] } [, ...] ) ] | GLOBAL ]
    [ INCLUDE ( column_name [, ...] )]
    [ WITH ( { storage_parameter = value } [, ...] ) ]
    [ TABLESPACE tablespace_name ]
    [ WHERE predicate ];
```

Parameter Description

- **UNIQUE**

  Creates a unique index. In this way, the system checks whether new values are unique in the index column. Attempts to insert or update data which would result in duplicate entries will generate an error.

  Currently, only B-tree indexes and UB-tree indexes support unique indexes.

- **CONCURRENTLY**

  Create an index (with ShareUpdateExclusiveLock) in non-blocking DML mode. A normal **CREATE INDEX** acquires exclusive lock on the table on which the index depends, blocking other accesses until the index drop can be completed. If this keyword is specified, DML is not blocked during the creation.

  – This option can only specify a name of one index.

  – The **CREATE INDEX** statement can be run within a transaction, but **CREATE INDEX CONCURRENTLY** cannot.

  – Column-store tables and partitioned tables do not support **CREATE INDEX CONCURRENTLY**. For temporary tables, you can use **CONCURRENTLY** to create indexes. However, indexes are created in blocking mode because no other sessions concurrently access the temporary tables and the blocking mode is more cost-effective.

  NOTE

  - This keyword is specified when an index is created. The entire table needs to be scanned twice and built. When the table is scanned for the first time, an index is created and the read and write operations are not blocked. During the second scan, changes that have occurred since the first scan are merged and updated.
  - The table needs to be scanned and built twice, and all existing transactions that may modify the table must be completed. This means that the creation of the index takes a longer time than normal. In addition, the CPU and I/O consumption also affects other services.
  - If an index build fails, it leaves an "unusable" index. This index is ignored by the query, but it still consumes the update overhead. In this case, you are advised to delete the index and try **CREATE INDEX CONCURRENTLY** again.
  - After the second scan, index creation must wait for any transaction that holds a snapshot earlier than the snapshot taken by the second scan to terminate. In addition, the ShareUpdateExclusiveLock (level 4) added during index creation conflicts with a lock whose level is greater than or equal to 4. Therefore, when such an index is created, the system is prone to hang or deadlock. Example:
    - If two sessions create an index by using **CONCURRENTLY** for the same table, a deadlock occurs.
    - If a session creates an index by using **CONCURRENTLY** for a table and another session drops a table, a deadlock occurs.
    - There are three sessions. Session 1 locks table **a** and does not commit it. Session 2 creates an index by using **CONCURRENTLY** for table **b**. Session 3 writes data to table **a**. Before the transaction of session 1 is committed, session 2 is blocked.
    - The transaction isolation level is set to repeatable read (read committed by default). Two sessions are started. Session 1 writes data to table **a** and does not commit it. Session 2 creates an index by using **CONCURRENTLY** for table **b**. Before the transaction of session 1 is committed, session 2 is blocked.

- **schema_name**

  Specifies the schema name.

Value range: an existing schema name

- **index_name**

  Specifies the name of the index to be created. The schema of the index is the same as that of the table.

  Value range: a string. It must comply with the identifier naming convention.

- **table_name**

  Specifies the name of the table to be indexed (optionally schema-qualified).

  Value range: an existing table name

- **USING method**

  Specifies the name of the index method to be used.

  Value range:

  – **btree**: B-tree indexes store key values of data in a B+ tree structure. This structure helps users to quickly search for indexes. B-tree indexes support comparison queries with ranges specified.

  – **gin**: GIN indexes are reverse indexes and can process values that contain multiple keys (for example, arrays).

  – **gist**: GiST indexes are suitable for the set data type and multidimensional data types, such as geometric and geographic data types. The following data types are supported: box, point, poly, circle, tsvector, tsquery, and range.

  – **Psort**: psort index. It is used to perform partial sort on column-store tables.

  – **ubtree**: Multi-version B-tree index used only for Ustore tables. The index page contains transaction information and can be recycled. By default, the INSERTPT function is enabled for UB-tree indexes.

  Row-store tables (Astore storage engine) support the following index types: **btree** (default), **gin**, and **gist**. Row-store tables (Ustore storage engine) support the following index type: **ubtree**. Column-store tables support the following index types: **Psort** (default), **btree**, and **gin**. Global temporary tables do not support GIN and GiST indexes.

  NOTE

  Column-store tables support GIN indexes only for the tsvector type. That is, the input parameter for creating a column-store GIN index must be the return value of the **to_tsvector** function. This method is commonly used for GIN indexes.

- **column_name**

  Specifies the name of the column on which an index is to be created.

  Multiple columns can be specified if the index method supports multi-column indexes. A global index supports a maximum of 31 columns, and other indexes support a maximum of 32 columns.

- **expression**

  Specifies an expression based on one or more columns of the table. The expression usually must be written with surrounding parentheses, as shown in the syntax. However, the parentheses can be omitted if the expression has the form of a function call.

  Expression can be used to obtain fast access to data based on some transformation of the basic data. For example, an index computed on

upper(col) would allow the clause **WHERE upper(col) = 'JIM'** to use an index.

If an expression contains **IS NULL**, the index for this expression is invalid. In this case, you are advised to create a partial index.

- **COLLATE collation**

  Assigns a collation to the column (which must be of a collatable data type). If no collation is specified, the default collation is used. You can run the **select * from pg_collation** command to query collation rules from the **pg_collation** system catalog. The default collation rule is the row starting with **default** in the query result.

- **opclass**

  Specifies the name of an operator class. An operator class can be specified for each column of an index. The operator class identifies the operators to be used by the index for that column. For example, a B-tree index on the type int4 would use the **int4_ops** class; this operator class includes comparison functions for values of type int4. In practice, the default operator class for the column's data type is sufficient. The operator class applies to data with multiple sorts. For example, users might want to sort a complex-number data type either by absolute value or by real part. They could do this by defining two operator classes for the data type and then selecting the proper class when making an index.

- **ASC**

  Specifies an ascending (default) sort order.

- **DESC**

  Specifies a descending sort order.

- **NULLS FIRST**

  Specifies that null values appear before non-null values in the sort ordering. This is the default when **DESC** is specified.

- **NULLS LAST**

  Specifies that null values appear after non-null values in the sort ordering. This is the default when **DESC** is not specified.

- **LOCAL**

  Specifies that the partitioned index to be created is a local index.

- **GLOBAL**

  Specifies the partitioned index to be created as a global index. If no keyword is specified, a global index is created by default.

- **INCLUDE (column_name [, ...]**)

  The optional **INCLUDE** clause specifies that some non-key columns are included in indexes. Non-key columns cannot be used as search criteria for accelerating index scans, and they are omitted when the unique constraints of the indexes are checked.

  An index-only scan can directly return content in the non-key columns without accessing the heap table corresponding to the indexes.

  Exercise caution when adding non-key columns as **INCLUDE** columns, especially for wide columns. If the size of an index tuple exceeds the maximum size allowed by the index type, data insertion fails. Note that in any case, adding non-key columns to an index increases the space occupied by the index, which may slow down the search speed.

Currently, only UB-tree indexes access mode supports this feature. Non-key columns are stored in the index leaf tuple corresponding to the heap tuple and are not included in the tuple on the upper-layer index page.

- **WITH ({storage_parameter = value} [, ...])**

  Specifies the storage parameter used for an index.

  Value range:

  Only index GIN supports parameters **FASTUPDATE** and **GIN_PENDING_LIST_LIMIT**. Indexes other than GIN and psort support the **FILLFACTOR** parameter. Only UB-tree indexes support **INDEXSPLIT**.

  – FILLFACTOR

    The fill factor of an index is a percentage from 10 to 100.

    Value range: 10–100

  – FASTUPDATE

    Specifies whether fast update is enabled for the GIN index.

    Value range: : **ON** and **OFF**

    Default value: **ON**

  – GIN_PENDING_LIST_LIMIT

    Specifies the maximum capacity of the pending list of the GIN index when fast update is enabled for the GIN index.

    Value range: 64–*INT_MAX*. The unit is KB.

    Default value: The default value of **gin_pending_list_limit** depends on **gin_pending_list_limit** specified in GUC parameters. By default, the value is 4 MB.

  – INDEXSPLIT

    Specifies the splitting policy of UB-tree indexes. The **DEFAULT** policy is the same as the splitting policy of B-tree indexes. The **INSERTPT** policy can significantly reduce the index space usage in some scenarios.

    Value range: **INSERTPT** and **DEFAULT**

    Default value: **INSERTPT**

- **TABLESPACE tablespace_name**

  Specifies the tablespace for an index. If no tablespace is specified, the default tablespace is used.

  Value range: an existing table name

- **WHERE predicate**

  Creates a partial index. A partial index is an index that contains entries for only a portion of a table, usually a portion that is more useful for indexing than the rest of the table. For example, if you have a table that contains both billed and unbilled orders where the unbilled orders take up a small fraction of the total table and yet that is an often used portion, you can improve performance by creating an index on just that portion. In addition, **WHERE** with **UNIQUE** can be used to enforce uniqueness over a subset for a table.

  Value range: The **predicate** expression can only refer to columns of the underlying table, but it can use all columns, not just the ones being indexed.

Currently, subqueries and aggregate expressions are forbidden in **WHERE**. You are not advised to use numeric types such as int for **predicate**, because such types can be implicitly converted to bool values (non-zero values are implicitly converted to **true** and **0** is implicitly converted to **false**), which may cause unexpected results.

For a partitioned table index, if the created index contains the GLOBAL or LOCAL keyword or the created index is a GLOBAL index, the WHERE clause cannot be used to create an index.

- **PARTITION index_partition_name**

  Specifies the name of an index partition.

  Value range: a string. It must comply with the identifier naming convention.

- **SUBPARTITION index_subpartition_name**

  Specifies the name of a level-2 index partition.

  Value range: a string. It must comply with the identifier naming convention.

- **TABLESPACE index_partition_tablespace**

  Specifies the tablespace of an index partition.

  Value range: If this parameter is not specified, the value of **index_tablespace** is used.

Examples

```
-- Create the tpcds.ship_mode_t1 table.
openGauss=# create schema tpcds;
openGauss=# CREATE TABLE tpcds.ship_mode_t1
(
    SM_SHIP_MODE_SK           INTEGER               NOT NULL,
    SM_SHIP_MODE_ID           CHAR(16)              NOT NULL,
    SM_TYPE                   CHAR(30)                      ,
    SM_CODE                   CHAR(10)                      ,
    SM_CARRIER                CHAR(20)                      ,
    SM_CONTRACT               CHAR(20)
)
;

-- Create a common unique index on the SM_SHIP_MODE_SK column in the tpcds.ship_mode_t1 table.
openGauss=# CREATE UNIQUE INDEX ds_ship_mode_t1_index1 ON
tpcds.ship_mode_t1(SM_SHIP_MODE_SK);

-- Create a B-tree index on the SM_SHIP_MODE_SK column in the tpcds.ship_mode_t1 table.
openGauss=# CREATE INDEX ds_ship_mode_t1_index4 ON tpcds.ship_mode_t1 USING
btree(SM_SHIP_MODE_SK);

-- Create an expression index on the SM_CODE column in the tpcds.ship_mode_t1 table:
openGauss=# CREATE INDEX ds_ship_mode_t1_index2 ON tpcds.ship_mode_t1(SUBSTR(SM_CODE,1 ,4));

-- Create a partial index on the SM_SHIP_MODE_SK column where SM_SHIP_MODE_SK is greater than 10
in the tpcds.ship_mode_t1 table.
openGauss=# CREATE UNIQUE INDEX ds_ship_mode_t1_index3 ON
tpcds.ship_mode_t1(SM_SHIP_MODE_SK) WHERE SM_SHIP_MODE_SK>10;

-- Rename an existing index.
openGauss=# ALTER INDEX tpcds.ds_ship_mode_t1_index1 RENAME TO ds_ship_mode_t1_index5;

-- Set the index as unusable.
openGauss=# ALTER INDEX tpcds.ds_ship_mode_t1_index2 UNUSABLE;

-- Rebuild an index.
openGauss=# ALTER INDEX tpcds.ds_ship_mode_t1_index2 REBUILD;

-- Delete an existing index.
```

```
openGauss=# DROP INDEX tpcds.ds_ship_mode_t1_index2;

-- Delete the table.
openGauss=# DROP TABLE tpcds.ship_mode_t1;

-- Create a tablespace.
openGauss=# CREATE TABLESPACE example1 RELATIVE LOCATION 'tablespace1/tablespace_1';
openGauss=# CREATE TABLESPACE example2 RELATIVE LOCATION 'tablespace2/tablespace_2';
openGauss=# CREATE TABLESPACE example3 RELATIVE LOCATION 'tablespace3/tablespace_3';
openGauss=# CREATE TABLESPACE example4 RELATIVE LOCATION 'tablespace4/tablespace_4';
-- Create the tpcds.customer_address_p1 table.
openGauss=# CREATE TABLE tpcds.customer_address_p1
(
    CA_ADDRESS_SK             INTEGER               NOT NULL,
    CA_ADDRESS_ID             CHAR(16)              NOT NULL,
    CA_STREET_NUMBER          CHAR(10)                      ,
    CA_STREET_NAME            VARCHAR(60)                   ,
    CA_STREET_TYPE            CHAR(15)                      ,
    CA_SUITE_NUMBER           CHAR(10)                      ,
    CA_CITY                   VARCHAR(60)                   ,
    CA_COUNTY                 VARCHAR(30)                   ,
    CA_STATE                  CHAR(2)                       ,
    CA_ZIP                    CHAR(10)                      ,
    CA_COUNTRY                VARCHAR(20)                   ,
    CA_GMT_OFFSET             DECIMAL(5,2)                  ,
    CA_LOCATION_TYPE          CHAR(20)
)
TABLESPACE example1
PARTITION BY RANGE(CA_ADDRESS_SK)
(
   PARTITION p1 VALUES LESS THAN (3000),
   PARTITION p2 VALUES LESS THAN (5000) TABLESPACE example1,
   PARTITION p3 VALUES LESS THAN (MAXVALUE) TABLESPACE example2
)
ENABLE ROW MOVEMENT;
-- Create the partitioned table index ds_customer_address_p1_index1 without specifying the index
partition name.
openGauss=# CREATE INDEX ds_customer_address_p1_index1 ON
tpcds.customer_address_p1(CA_ADDRESS_SK) LOCAL;
-- Create the partitioned table index ds_customer_address_p1_index2 with the name of the index partition
specified.
openGauss=# CREATE INDEX ds_customer_address_p1_index2 ON
tpcds.customer_address_p1(CA_ADDRESS_SK) LOCAL
(
    PARTITION CA_ADDRESS_SK_index1,
    PARTITION CA_ADDRESS_SK_index2 TABLESPACE example3,
    PARTITION CA_ADDRESS_SK_index3 TABLESPACE example4
)
TABLESPACE example2;

-- Create a global partitioned index.
openGauss=CREATE INDEX ds_customer_address_p1_index3 ON
tpcds.customer_address_p1(CA_ADDRESS_ID) GLOBAL;

-- If no keyword is specified, a global partitioned index is created by default.
openGauss=CREATE INDEX ds_customer_address_p1_index4 ON
tpcds.customer_address_p1(CA_ADDRESS_ID);

-- Change the tablespace of the partitioned table index CA_ADDRESS_SK_index2 to example1.
openGauss=# ALTER INDEX tpcds.ds_customer_address_p1_index2 MOVE PARTITION
CA_ADDRESS_SK_index2 TABLESPACE example1;

-- Change the tablespace of the partitioned table index CA_ADDRESS_SK_index3 to example2.
openGauss=# ALTER INDEX tpcds.ds_customer_address_p1_index2 MOVE PARTITION
CA_ADDRESS_SK_index3 TABLESPACE example2;

-- Rename a partitioned table index.
openGauss=# ALTER INDEX tpcds.ds_customer_address_p1_index2 RENAME PARTITION
CA_ADDRESS_SK_index1 TO CA_ADDRESS_SK_index4;
```

```
-- Delete the created indexes and the partitioned table.
openGauss=# DROP INDEX tpcds.ds_customer_address_p1_index1;
openGauss=# DROP INDEX tpcds.ds_customer_address_p1_index2;
openGauss=# DROP TABLE tpcds.customer_address_p1;
-- Delete the tablespace.
openGauss=# DROP TABLESPACE example1;
openGauss=# DROP TABLESPACE example2;
openGauss=# DROP TABLESPACE example3;
openGauss=# DROP TABLESPACE example4;

-- Create a column-store table and its GIN index:
openGauss=# create table cgin_create_test(a int, b text) with (orientation = column);
CREATE TABLE
openGauss=# create index cgin_test on cgin_create_test using gin(to_tsvector('ngram', b));
CREATE INDEX
```

Helpful Links

ALTER INDEX and **DROP INDEX**.

Suggestions

- create index

  You are advised to create indexes on:

  – Columns that are often queried

  – Join conditions. For a query on joined columns, you are advised to create a composite index on the columns. For example, for **select * from t1 join t2 on t1.a=t2.a and t1.b=t2.b**, you can create a composite index on columns **a** and **b** in table **t1**.

  – Columns having filter criteria (especially scope criteria) of a **where** clause

  – Columns that appear after **order by**, **group by**, and **distinct**

  Constraints:

  – An index of an ordinary table supports a maximum of 32 columns. A GLOBAL index of a partitioned table supports a maximum of 31 columns.

  – The size of a single index cannot exceed the size of the index page (8 KB). The size of a B-tree, UB-tree, or GIN index cannot exceed one third of the page size.

  – Partial indexes cannot be created in a partitioned table.

  – When a GLOBAL index is created on a partitioned table, the following constraints apply:

    ■ Expression indexes and partial indexes are not supported.

    ■ Row-store tables are not supported.

    ■ Only B-tree indexes are supported.

  – In the same attribute column, the local index and global index of a partition cannot coexist.

  – If the **ALTER** statement does not contain **UPDATE GLOBAL INDEX**, the original GLOBAL index is invalid. In this case, other indexes are used for query. If the ALTER statement contains UPDATE GLOBAL INDEX, the original GLOBAL index is still valid and the index function is correct.

11.14.69 CREATE LANGUAGE

This version does not support this syntax.

11.14.70 CREATE MASKING POLICY

Function

CREATE MASKING POLICY creates a masking policy.

Precautions

Only users with the **poladmin** or **sysadmin** permission, or the initial user can perform this operation.

The masking policy takes effect only after the security policy is enabled, that is, **enable_security_policy** is set to **on**. For details, see "Database Configuration > Database Security Management Policies > Dynamic Data Masking" in *Security Hardening Guide*.

For details about the execution effect and supported data types of preset masking functions, see "Database Security > Dynamic Data Masking" in *Feature Description*.

Syntax

```
CREATE MASKING POLICY policy_name masking_clause[, ...]* policy_filter [ENABLE | DISABLE];
```

- masking_clause
  ```
  masking_function ON LABEL(label_name[, ...]*)
  ```
- masking_function

  maskall is not a preset function. It is hard-coded and cannot be displayed by running **\df**.

  The masking methods during presetting are as follows:
  ```
  maskall | randommasking | creditcardmasking | basicemailmasking | fullemailmasking |
  shufflemasking | alldigitsmasking | regexpmasking
  ```
- policy_filter:
  ```
  FILTER ON FILTER_TYPE(filter_value [,...]*)[,...]*
  ```
- FILTER_TYPE:
  ```
  IP | APP | ROLES
  ```

Parameter Description

- **policy_name**

  Specifies the audit policy name, which must be unique.

  Value range: a string. It must comply with the naming convention.
- **label_name**

  Specifies the resource label name.
- **masking_clause**

  Specifies the masking function to be used to anonymize database resources labeled by **label_name**. **schema.function** can be used to specify the masking function.

- **policy_filter**

  Specifies the users for which the masking policy takes effect. If this parameter is left empty, the masking policy takes effect for all users.
- **FILTER_TYPE**

  Specifies the types of information to be filtered by the policy, including **IP**, **APP**, and **ROLES**.
- **filter_value**

  Indicates the detailed information to be filtered, such as the IP address, app name, and username.
- **ENABLE|DISABLE**

  Enables or disables the masking policy. If **ENABLE|DISABLE** is not specified, **ENABLE** is used by default.

Examples

```
-- Create users dev_mask and bob_mask.
openGauss=# CREATE USER dev_mask PASSWORD 'dev@1234';
openGauss=# CREATE USER bob_mask PASSWORD 'bob@1234';

-- Create table tb_for_masking.
openGauss=# CREATE TABLE tb_for_masking(col1 text, col2 text, col3 text);

-- Create a resource label for label sensitive column col1.
openGauss=# CREATE RESOURCE LABEL mask_lb1 ADD COLUMN(tb_for_masking.col1);

-- Create a resource label for label sensitive column col2.
openGauss=# CREATE RESOURCE LABEL mask_lb2 ADD COLUMN(tb_for_masking.col2);

-- Create a masking policy for the operation of accessing sensitive column col1.
openGauss=# CREATE MASKING POLICY maskpol1 maskall ON LABEL(mask_lb1);

-- Create a masking policy that takes effect only for scenarios where users are dev_mask and bob_mask,
client tools are psql and gsql, and IP addresses are 10.20.30.40, and 127.0.0.0/24.
openGauss=# CREATE MASKING POLICY maskpol2 randommasking ON LABEL(mask_lb2) FILTER ON
ROLES(dev_mask, bob_mask), APP(psql, gsql), IP('10.20.30.40', '127.0.0.0/24');
```

Helpful Links

ALTER MASKING POLICY and DROP MASKING POLICY

11.14.71 CREATE MATERIALIZED VIEW

CREATE MATERIALIZED VIEW creates a complete-refresh materialized view, and you can use **REFRESH MATERIALIZED VIEW** to fully refresh the data in the materialized view.

CREATE MATERIALIZED VIEW is similar to **CREATE TABLE AS**, but it remembers the query used to initialize the view, so it can refresh data later. A materialized view has many attributes that are the same as those of a table, but does not support temporary materialized views.

Precautions

- Complete-refresh materialized views cannot be created in temporary tables or global temporary tables.
- Complete-refresh materialized views do not support NodeGroups.

- After a complete-refresh materialized view is created, most DDL operations in the base table are no longer supported.
- The IUD operation cannot be performed on complete-refresh materialized views.
- After a complete-refresh materialized view is created, if the base table data changes, you need to run the **REFRESH** command to synchronize the materialized view with the base table.
- The Ustore engine does not support the creation and use of materialized views.

Syntax

```
CREATE MATERIALIZED VIEW mv_name
    [ (column_name [, ...] ) ]
    [ WITH ( {storage_parameter = value} [, ... ] ) ]
      [ TABLESPACE tablespace_name ]
    AS query
    [ WITH [ NO ] DATA ];
```

Parameter Description

- **mv_name**

  Name (optionally schema-qualified) of the materialized view to be created.

  Value range: a string. It must comply with the identifier naming convention.
- **column_name**

  Specifies a column name in the new materialized view. The materialized view supports specified columns. The number of specified columns must be the same as the number of columns in the result of the subsequent query statement. If no column name is provided, the column name is obtained from the output column name of the query.

  Value range: a string. It must comply with the identifier naming convention.
- **WITH (storage_parameter [= value] [, ...])**

  Specifies an optional storage parameter for a table or an index. For details, see **CREATE TABLE**.
- **TABLESPACE tablespace_name**

  Tablespace to which the new materialized view belongs. If not specified, the default tablespace is used.
- **AS query**

  Specifies the **SELECT**, **TABLE**, or **VALUES** command. This query will be run in a security-constrained operation.

Examples

```
-- Create an ordinary table.
openGauss=# CREATE TABLE my_table (c1 int, c2 int);
-- Create a complete-refresh materialized view.
openGauss=# CREATE MATERIALIZED VIEW my_mv AS SELECT * FROM my_table;
-- Write data to the base table.
openGauss=# INSERT INTO my_table VALUES(1,1),(2,2);
-- Refresh the complete-refresh materialized view my_mv.
openGauss=# REFRESH MATERIALIZED VIEW my_mv;
```

Helpful Links

ALTER MATERIALIZED VIEW, **CREATE INCREMENTAL MATERIALIZED VIEW**, **CREATE TABLE**, **DROP MATERIALIZED VIEW**, **REFRESH INCREMENTAL MATERIALIZED VIEW**, and **REFRESH MATERIALIZED VIEW**

11.14.72 CREATE MODEL

Function

CREATE MODEL trains a machine learning model and saves the model.

Precautions

- The model name must be unique. Pay attention to the naming format.
- The AI training duration fluctuates greatly, and in some cases, the training duration is long. If the duration specified by the GUC parameter **statement_timeout** is too long, the training will be interrupted. You are advised to set **statement_timeout** to **0** so that the statement execution duration is not limited.

Syntax

```
CREATE MODEL model_name USING algorithm_name
[FEATURES { {expression [ [ AS ] output_name ]} [, ...] }]
[TARGET { {expression [ [ AS ] output_name ]} [, ...] }]
FROM { table_name | select_query }
WITH hyperparameter_name = { hyperparameter_value | DEFAULT } [, ...] }
```

Parameter Description

- **model_name**

  Name of the training model, which must be unique.

  Value range: a string. It must comply with the identifier naming convention.
- **architecture_name**

  Algorithm type of the training model.

  Value range: a string. Currently, the value can be **logistic_regression**, **linear_regression**, **svm_classification**, or **kmeans**.
- **attribute_list**

  Enumerated input column name of the training model.

  Value range: a string. It must comply with the naming convention of data attributes.
- **attribute_name**

  Target column name of the retraining model in a supervised learning task (simple expression processing can be performed).

  Value range: a string. It must comply with the naming convention of data attributes.
- **subquery**

  Data source.

  Value range: a string. It must comply with the SQL syntax of databases.

Examples

```
CREATE TABLE houses (
id INTEGER,
tax INTEGER,
bedroom INTEGER,
bath DOUBLE PRECISION,
price INTEGER,
size INTEGER,
lot INTEGER,
mark text
);
insert into houses(id, tax, bedroom, bath, price, size, lot, mark) VALUES
(1,590,2,1,50000,770,22100,'a+'),
(2,1050,3,2,85000,1410,12000,'a+'),
(3,20,2,1,22500,1060,3500,'a-'),
(4,870,2,2,90000,1300,17500,'a+'),
(5,1320,3,2,133000,1500,30000,'a+'),
(6,1350,2,1,90500,850,25700,'a-'),
(7,2790,3,2.5,260000,2130,25000,'a+'),
(8,680,2,1,142500,1170,22000,'a-'),
(9,1840,3,2,160000,1500,19000,'a+'),
(10,3680,4,2,240000,2790,20000,'a-'),
(11,1660,3,1,87000,1030,17500,'a+'),
(12,1620,3,2,118500,1250,20000,'a-'),
(13,3100,3,2,140000,1760,38000,'a+'),
(14,2090,2,3,148000,1550,14000,'a-'),
(15,650,3,1.5,65000,1450,12000,'a-');
CREATE MODEL price_model USING logistic_regression
 FEATURES size, lot
 TARGET mark
 FROM HOUSES
 WITH learning_rate=0.88, max_iterations=default;
```

Helpful Links

DROP MODEL and **PREDICT BY**

11.14.73 CREATE OPERATOR

Function

CREATE OPERATOR defines a new operator.

Precautions

CREATE OPERATOR defines a new name operator. The user who defines the operator becomes the owner of the operator. If a schema name is given, the operator is created in the specified schema. Otherwise, it will be created in the current schema.

The operator name is a character string consisting of the following characters:

+ - * / < > = ~ ! @ # % ^ & | ` ?

When selecting a name, note the following restrictions:

- -- and /* cannot appear anywhere in the operator name, because they are regarded as the beginning of a comment.
- A multi-character operator cannot end with + or - unless the name contains at least one of the following characters:

  ~ ! @ # % ^ & | ` ?

- => The operator name is no longer used.

Operator! = is mapped to <> when being entered. Therefore, the two names are always equivalent.

At least one LEFTARG and one RIGHTARG must be defined. For binocular operators, both need to be defined. For the right operator, only LEFTARG needs to be defined. For the left operator, only RIGHTARG needs to be defined.

Also, the function_name procedure must have been defined with CREATE FUNCTION, and must be defined to accept the correct number of specified type parameters (one or two).

Other clauses declare optional operator optimization clauses. Their meanings are defined in **Section 35.13**.

To create an operator, you must have the USAGE permission on the parameter type and return type, and the EXECUTE permission on the underlying function. If exchange or negative operators are specified, you must have them.

Syntax

```
CREATE OPERATOR name (
    PROCEDURE = function_name
    [, LEFTARG = left_type ] [, RIGHTARG = right_type ]
    [, COMMUTATOR = com_op ] [, NEGATOR = neg_op ]
    [, RESTRICT = res_proc ] [, JOIN = join_proc ]
    [, HASHES ] [, MERGES ]
)
```

Parameter Description

- **name**

  Operator to be defined. The available characters are listed above. The name can be schema-qualified, for example, CREATE OPERATOR myschema.+ (...). If there is no schema, the operator is created in the current schema. Two operators in the same schema can have the same name as long as they operate on different data types. This is a reloading process.

- **function_name**

  Function used to implement the operator.

- **left_type**

  Parameter data type on the left of the operator, if any. This parameter can be omitted if the left operator is used.

- **right_type**

  Parameter data type on the right of the operator, if any. This parameter can be omitted if the right-view operator is used.

- **com_op**

  Exchange operator corresponding to the operator.

- **neg_op**

  Negative operator corresponding to the operator.

- **res_proc**

  This operator constrains the selectivity evaluation function.

- **join_proc**

  This operator joins the selectivity evaluation function.
- **HASHES**

  Indicates that the operator supports hash joins.
- **MERGES**

  Indicates that this operator supports a merge join.

  Use the OPERATOR() syntax to provide a schema-qualified operator name in com_op or other optional parameters. For example:

  ```
  COMMUTATOR = OPERATOR(myschema.===) ,
  ```

Example

The following command defines a new operator: equal area for the box data type.

```
CREATE OPERATOR === (
    LEFTARG = box,
    RIGHTARG = box,
    PROCEDURE = area_equal_procedure,
    COMMUTATOR = ===,
    NEGATOR = !==,
    RESTRICT = area_restriction_procedure,
    JOIN = area_join_procedure,
    HASHES, MERGES
);
```

11.14.74 CREATE PACKAGE

Function

CREATE PACKAGE creates a package.

Precautions

- The package can be used only in centralized databases and cannot be used in distributed databases.
- The functions or stored procedures declared in the package specification must be defined in the package body.
- During instantiation, the stored procedure with **commit** or **rollback** cannot be invoked.
- Package functions cannot be invoked in triggers.
- Variables in a package cannot be directly used in external SQL statements.
- Private variables and stored procedures in a package cannot be invoked outside the package.
- Usage that other stored procedures do not support are not supported. For example, if **commit** or **rollback** cannot be invoked in a function, **commit** or **rollback** cannot be invoked in the function of a package.
- The name of a schema cannot be the same as that of a package.
- Only A-version stored procedures and function definitions are supported.
- Variables with the same name in a package, including parameters with the same name in a package, are not supported.
- The global variables in a package are at the session level. The variables in packages cannot be shared in different sessions.

- When a function of an autonomous transaction is called in a package, the cursor variables in the package and recursive functions that use the cursor variables in the package are not allowed.
- The package does not declare the ref cursor variables.
- The default permission on a package is **SECURITY INVOKER**. To change the default permission to **SECURITY DEFINER**, set the GUC parameter **behavior_compat_options** to **'plsql_security_definer'**.
- A user granted with the **CREATE ANY PACKAGE** permission can create packages in the public and user schemas.
- If the name of a package to be created contains special characters, the special characters cannot contain spaces. You are advised to set the GUC parameter **behavior_compat_options** to **"skip_insert_gs_source"**. Otherwise, an error may occur.

Syntax

- **CREATE PACKAGE SPECIFICATION**

```
CREATE [ OR REPLACE ] PACKAGE [ schema ] package_name
    [ invoker_rights_clause ] { IS | AS } item_list_1 END package_name;

invoker_rights_clause can be declared as AUTHID DEFINER or AUTHID INVOKER, which indicate
the definer permission and invoker permission, respectively.
item_list_1 can be a declared variable, stored procedure, or function.
```

  The package specification declares public variables, functions, and exceptions in a package, which can be invoked by external functions or stored procedures. It can only declare stored procedures and functions but cannot define them.

- **CREATE PACKAGE BODY**

```
CREATE [ OR REPLACE ] PACKAGE BODY [ schema ] package_name
    { IS | AS } declare_section [ initialize_section ] END package_name;
```

  The package body defines private variables and functions in a package. If a variable or function is not declared by the package specification, it is a private variable or function.

  The package body also has an initialization part to initialize the package. For details, see the example.

Examples

- Example of **CREATE PACKAGE SPECIFICATION**

```
CREATE OR REPLACE PACKAGE emp_bonus IS
var1 int:=1;-- Public variable
var2 int:=2;
PROCEDURE testpro1(var3 int);-- Public stored procedure, which can be called by external systems.
END emp_bonus;
/
```

- Example of **CREATE PACKAGE BODY**

```
drop table if exists test1;
create or replace package body emp_bonus is
var3 int:=3;
var4 int:=4;
procedure testpro1(var3 int)
is
begin
create table if not exists test1(col1 int);
insert into test1 values(var1);
insert into test1 values(var4);
end;
```

```
begin: --The instantiation starts.
var4:=9;
testpro1(var4);
end emp_bonus;
/
```

- Example of **ALTER PACKAGE OWNER**

```
ALTER PACKAGE emp_bonus OWNER TO omm;
-- Change the owner of PACKAGE emp_bonus to omm.
```

- Example of calling a package

```
call emp_bonus.testpro1(1); -- Use call to call the stored procedure of a package.
select emp_bonus.testpro1(1); -- Use select to call the stored procedure of a package.
-- Call the stored procedure of a package in an anonymous block.
begin
emp_bonus.testpro1(1);
end;
/
```

11.14.75 CREATE PROCEDURE

Function

CREATE PROCEDURE creates a stored procedure.

Precautions

- If the parameters or return values of a stored procedure have precision, the precision is not checked.
- When creating a stored procedure, you are advised to explicitly specify the schemas of all operations on table objects in the stored procedure definition. Otherwise, the stored procedure may fail to be executed.
- **current_schema** and **search_path** specified by **SET** during stored procedure creation are invalid. **search_path** and **current_schema** before and after function execution should be the same.
- If a stored procedure has output parameters, the **SELECT** statement uses the default values of the output parameters when calling the procedure. When the **CALL** statement calls the stored procedure or a non-overloaded function, output parameters must be specified. When the **CALL** statement calls an overloaded **PACKAGE** function, it can use the default values of the output parameters. For details, see examples in **CALL**.
- A stored procedure with the **PACKAGE** attribute can use overloaded functions.
- When you create a procedure, you cannot insert aggregate functions or other functions out of the average function.
- When stored procedures without parameters are called in another stored procedure, you can omit brackets and call stored procedures using their names directly.
- When functions with output parameters are called in a stored procedure which is an assignment expression, you can omit the output parameters of the called functions.
- The stored procedure supports viewing, exporting, and importing parameter comments.
- The stored procedure supports viewing, exporting, and importing parameter comments between IS/AS and plsql_body.

- The default permission on a stored procedure is **SECURITY INVOKER**. If you want to change the default permission to **SECURITY DEFINER**, you need to set the GUC parameter **behavior_compat_options** to **'plsql_security_definer'**. For details about the S**ECURITY DEFINER** permission, see section "Permission Control" in *Security Hardening Guide*.
- Users granted with the **CREATE ANY FUNCTION** permission can create or replace stored procedures in the user schemas.
- **out/inout** must be set to a variable but not a constant.
- In a centralized environment, if you want to call a stored procedure with the same in parameters but different out parameters, you need to set the GUC parameter **behavior_compat_options** to **'proc_outparam_override'**. After the parameter is enabled, you must add the out parameters no matter whether you use the SELECT or CALL statement to call the stored procedure. After the parameter is enabled, you cannot use **perform** to call a stored procedure or function.

Syntax

```
CREATE [ OR REPLACE ] PROCEDURE procedure_name
    [ ( {[ argname ] [ argmode ] argtype [ { DEFAULT | := | = } expression ]}[,...]) ]
    [
       { IMMUTABLE | STABLE | VOLATILE }
       | { SHIPPABLE | NOT SHIPPABLE }
       | {PACKAGE}
       | [ NOT ] LEAKPROOF
       | { CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT }
       | {[ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER | AUTHID DEFINER | AUTHID 
CURRENT_USER}
       | COST execution_cost
       | SET configuration_parameter { TO value | = value | FROM CURRENT }
    ][ ... ]
 { IS | AS }
plsql_body
/
```

Parameter Description

- **OR REPLACE**

  Replaces the original definition when two stored procedures are with the same name.
- **procedure_name**

  Specifies the name of the stored procedure that is created (optionally with schema names).

  Value range: a string. It must comply with the identifier naming convention.
- **argmode**

  Specifies the mode of an argument.

  NOTICE

  VARIADIC specifies parameters of the array type.

  Value range: **IN**, **OUT**, **INOUT**, and **VARIADIC**. The default value is **IN**. Only the parameters in **OUT** mode can follow the **VARIADIC** parameter.

- **argname**

  Specifies the argument name.

  Value range: a string. It must comply with the identifier naming convention.

- **argtype**

  Specifies the type of an argument. **%TYPE** or **%ROWTYPE** can be used to indirectly reference a variable or table type. For details, see **Variable Definition Statements**.

  Value range: a valid data type

  NOTE

  In **PROCEDURE argtype** outside **PACKAGE**, **%TYPE** cannot reference the type of the *PACKAGE* variable.

- **configuration_parameter**
  - **value**

    Sets the specified configuration parameter to a specified value. If the **value** is **DEFAULT**, the default setting is used in the new session. **OFF** disables the setting.

    Value range: a string

    - DEFAULT
    - OFF
    - Specified default value
  - **from current**

    Uses the value of **configuration_parameter** of the current session.

- **IMMUTABLE, STABLE,...**

  Specifies a constraint. The function of each parameter is similar to that of **CREATE FUNCTION**. For details, see **CREATE FUNCTION**.

- **plsql_body**

  Specifies the PL/SQL stored procedure body.

NOTICE

When you create a user, or perform other operations requiring password input in a stored procedure, the system catalog and CSV log record the password in plaintext. Therefore, you are advised not to perform such operations in the stored procedure.

NOTE

No specific order is applied to **argname** and **argname**. The following order is advised: **argname**, **argmode**, and **argtype**.

Helpful Links

DROP PROCEDURE

Suggestions

- analyse | analyze
  - Do not run **ANALYZE** in a transaction or anonymous block.
  - Do not run **ANALYZE** in a function or stored procedure.

11.14.76 CREATE PUBLICATION

Function

CREATE PUBLICATION adds a new publication to the current database. The publication name must be different from the name of any existing publication in the current database. A publication is essentially the replication of data changes in a set of tables achieved by logical replication.

Precautions

- If neither **FOR TABLE** nor **FOR ALL TABLES** is specified, a publication starts with a set of empty tables. Tables can be added later.
- Creating a publication does not start replication. It defines only one group and filtering logic for future subscribers. To create a publication, the caller must have the **CREATE** permission on the current database. (The system administrator does not need to perform a check on this.)
- To add a table to a publication, the caller must have ownership of the table. The **FOR ALL TABLES** clause requires that the caller be a user with the **SYSADMIN** permission.
- Tables in the internal schemas of the database, including **blockchain**, **cstore**, **db4ai**, **dbe_pldebugger**, **dbe_pldeveloper**, **pkg_service**, **snapshot**, and **sqladvisor**, are not published.
- Tables added to a publication that publishes UPDATE or DELETE operations must already have **REPLICA IDENTITY** defined or have a primary key; otherwise, these operations will be prohibited in those tables.
- The **COPY... FROM** command is used to publish INSERT operations. It cannot be used to publish TRUNCATE and DDL operations.

Syntax

```
CREATE PUBLICATION name
    [ FOR TABLE table_name [, ...]
      | FOR ALL TABLES ]
    [ WITH ( publication_parameter [=value] [, ... ] ) ];
```

Parameter Description

- **name**

  Specifies the name of a new publication.
- **FOR TABLE**

  Specifies the list of tables to be added to a publication. Only persistent base tables can be published. Temporary tables, unlogged tables, foreign tables, MOTs, materialized views, and regular views cannot be published.
- **FOR ALL TABLES**

Marks a publication as replicating changes of all tables in the database, including tables to be created.

- **WITH (publication_parameter [= value] [, ...])**

  Specifies the optional parameters for a publication. The following parameters are supported:

  – **publish (string)**

    Specifies which DML operations can be published to subscribers. The value of this parameter is a list of operations separated by commas (,). The allowed operations are INSERT, UPDATE, and DELETE. If this parameter is not specified, all operations are published by default. The default value is **'insert, update, delete'**.

Examples

```
-- Create a publication to publish all changes in two tables.
CREATE PUBLICATION mypublication FOR TABLE users, departments;
-- Create a publication to publish all changes in all tables.
CREATE PUBLICATION alltables FOR ALL TABLES;
-- Create a publication to publish INSERT operations in only one table.
CREATE PUBLICATION insert_only FOR TABLE mydata WITH (publish = 'insert');
-- Modify publication operations.
ALTER PUBLICATION insert_only SET (publish='insert,update,delete');
-- Add a table to a publication.
ALTER PUBLICAITON insert_only ADD TABLE mydata2;
-- Delete a publication.
DROP PUBLICATION insert_only;
```

Helpful Links

ALTER PUBLICATION and **DROP PUBLICATION**

11.14.77 CREATE RESOURCE LABEL

Function

CREATE RESOURCE LABEL creates a resource label.

Precautions

Only users with the **poladmin** or **sysadmin** permission, or the initial user can perform this operation.

Syntax

```
CREATE RESOURCE LABEL [IF NOT EXISTS] label_name ADD label_item_list[, ...]*;
```

- label_item_list
  ```
  resource_type(resource_path[, ...]*)
  ```
- resource_type
  ```
  TABLE | COLUMN | SCHEMA | VIEW | FUNCTION
  ```

Parameter Description

- **label_name**

  Specifies the resource label name, which must be unique.

  Value range: a string. It must comply with the naming convention.

- **resource_type**

  Specifies the type of database resources to be labeled.
- **resource_path**

  Specifies the path of database resources.

Examples

```
-- Create table tb_for_label.
openGauss=# CREATE TABLE tb_for_label(col1 text, col2 text, col3 text);

-- Create schema schema_for_label.
openGauss=# CREATE SCHEMA schema_for_label;

-- Create view view_for_label.
openGauss=# CREATE VIEW view_for_label AS SELECT 1;

-- Create function func_for_label.
openGauss=# CREATE FUNCTION func_for_label RETURNS TEXT AS $$ SELECT col1 FROM tb_for_label; $$
LANGUAGE SQL;

-- Create a resource label based on the table.
openGauss=# CREATE RESOURCE LABEL IF NOT EXISTS table_label add TABLE(public.tb_for_label);

-- Create a resource label based on the columns.
openGauss=# CREATE RESOURCE LABEL IF NOT EXISTS column_label add
COLUMN(public.tb_for_label.col1);

-- Create a resource label based on the schema.
openGauss=# CREATE RESOURCE LABEL IF NOT EXISTS schema_label add SCHEMA(schema_for_label);

-- Create a resource label based on the view.
openGauss=# CREATE RESOURCE LABEL IF NOT EXISTS view_label add VIEW(view_for_label);

-- Create a resource label based on the function.
openGauss=# CREATE RESOURCE LABEL IF NOT EXISTS func_label add FUNCTION(func_for_label);
```

Helpful Links

ALTER RESOURCE LABEL and **DROP RESOURCE LABEL**

11.14.78 CREATE ROLE

Function

CREATE ROLE creates a role.

A role is an entity that owns database objects and permissions. In different environments, a role can be considered a user, a group, or both.

Precautions

- **CREATE ROLE** adds a role to a database. The role does not have the LOGIN permission.
- Only the user who has the **CREATE ROLE** permission or a system administrator is allowed to create roles.

Syntax

```
CREATE ROLE role_name [ [ WITH ] option [ ... ] ] [ ENCRYPTED | UNENCRYPTED ] { PASSWORD |
IDENTIFIED BY } { 'password' [EXPIRED] | DISABLE };
```

The syntax of role information configuration clause **option** is as follows:

```
{SYSADMIN | NOSYSADMIN}
    | {MONADMIN | NOMONADMIN}
    | {OPRADMIN | NOOPRADMIN}
    | {POLADMIN | NOPOLADMIN}
    | {AUDITADMIN | NOAUDITADMIN}
    | {CREATEDB | NOCREATEDB}
    | {USEFT | NOUSEFT}
    | {CREATEROLE | NOCREATEROLE}
    | {INHERIT | NOINHERIT}
    | {LOGIN | NOLOGIN}
    | {REPLICATION | NOREPLICATION}
    | {INDEPENDENT | NOINDEPENDENT}
    | {VCADMIN | NOVCADMIN}
    | {PERSISTENCE | NOPERSISTENCE}
    | CONNECTION LIMIT connlimit
    | VALID BEGIN 'timestamp'
    | VALID UNTIL 'timestamp'
    | RESOURCE POOL 'respool'
    | USER GROUP 'groupuser'
    | PERM SPACE 'spacelimit'
    | TEMP SPACE 'tmpspacelimit'
    | SPILL SPACE 'spillspacelimit'
    | NODE GROUP logic_cluster_name
    | IN ROLE role_name [, ...]
    | IN GROUP role_name [, ...]
    | ROLE role_name [, ...]
    | ADMIN rol e_name [, ...]
    | USER role_name [, ...]
    | SYSID uid
    | DEFAULT TABLESPACE tablespace_name
    | PROFILE DEFAULT
    | PROFILE profile_name
    | PGUSER
```

Parameter Description

- **role_name**

  Specifies the name of a role.

  Value range: a string. It must comply with the identifier naming convention and can contain a maximum of 63 characters. If the value contains more than 63 characters, the database truncates it and retains the first 63 characters as the role name. When a role is created, the database will display a message.

  NOTE

  The identifier must be letters, underscores (_), digits (0-9), or dollar signs ($) and must start with a letter (a-z) or underscore (_).

- **password**

  Specifies the login password.

  A new password must:

  – Contain at least eight characters. This is the default length.

  – Differ from the username or the username spelled backward.

  – Contain at least three of the following character types: uppercase characters, lowercase characters, digits, and special characters (limited to ~!@#$ %^&*()-_=+\|[{}];:,<.>/?).

  – The password can also be a ciphertext character string that meets the format requirements. This mode is mainly used to import user data. You are not advised to use it directly. If a ciphertext password is used, the user

  must know the plaintext corresponding to the ciphertext password and ensure that the plaintext password meets the complexity requirements. The database does not verify the complexity of the ciphertext password. Instead, the security of the ciphertext password is ensured by the user.
  - Be enclosed by single quotation marks when a role is created.

  Value range: a character string that cannot be empty.

- **EXPIRED**

  When creating a user, you can specify the **EXPIRED** parameter to create a user whose password is invalid. The user cannot perform simple or extended queries. The statement can be executed only after the password is changed.

- **DISABLE**

  By default, you can change your password unless it is disabled. To disable the password of a user, use this parameter. After the password of a user is disabled, the password will be deleted from the system. The user can connect to the database only through external authentication, for example, Kerberos authentication. Only administrators can enable or disable a password. Common users cannot disable the password of an initial user. To enable a password, run **ALTER USER** and specify the password.

- **ENCRYPTED | UNENCRYPTED**

  Controls whether the password is stored encrypted in the system catalogs. According to product security requirement, the password must be stored encrypted. Therefore, **UNENCRYPTED** is forbidden in GaussDB. If the password string has already been encrypted in the SHA256 format, it is stored encrypted as it was, regardless of whether **ENCRYPTED** or **UNENCRYPTED** is specified (since the system cannot decrypt the specified encrypted password string). This allows reloading of encrypted passwords during dump/restore.

- **SYSADMIN | NOSYSADMIN**

  Specifies whether a new role is a system administrator. Roles with the SYSADMIN attribute have the highest permission.

  Value range: If not specified, **NOSYSADMIN** is the default.

  When separation of duties is disabled, users with the SYSADMIN permission can create users with the SYSADMIN, REPLICATION, CREATEROLE, AUDITADMIN, MONADMIN, POLADMIN, or CREATEDB permission and common users.

  When separation of duties is enabled, users with the SYSADMIN permission do not have the permission to create users.

- **MONADMIN | NOMONADMIN**

  Specifies whether a role is a monitor administrator.

  Value range: If not specified, **NOMONADMIN** is the default.

- **OPRADMIN | NOOPRADMIN**

  Specifies whether a role is an O&M administrator.

  Value range: If not specified, **NOOPRADMIN** is the default.

- **POLADMIN | NOPOLADMIN**

  Specifies whether a role is a security policy administrator.

  Value range: If not specified, **NOPOLADMIN** is the default.

- **AUDITADMIN | NOAUDITADMIN**

Specifies whether a role has the audit and management attributes.

If not specified, **NOAUDITADMIN** is the default.

- **CREATEDB | NOCREATEDB**

  Specifies a role's permission to create databases.

  A new role does not have the permission to create databases.

  Value range: If not specified, **NOCREATEDB** is the default.

- **USEFT | NOUSEFT**

  This parameter is reserved and not used in this version.

- **CREATEROLE | NOCREATEROLE**

  Specifies whether a role will be permitted to create new roles (that is, execute **CREATE ROLE** and **CREATE USER**). A role with the CREATEROLE permission can also modify and delete other roles.

  Value range: If not specified, **NOCREATEROLE** is the default.

  When separation of duties is disabled, users with the CREATEROLE permission can create users with the CREATEROLE, AUDITADMIN, MONADMIN, POLADMIN, or CREATEDB permission and common users.

  When separation of duties is enabled, users with the CREATEROLE permission can create users with the CREATEROLE, MONADMIN, POLADMIN, or CREATEDB permission and common users.

- **INHERIT | NOINHERIT**

  Specifies whether a role "inherits" the permissions of roles in the same group. It is not recommended.

- **LOGIN | NOLOGIN**

  Specifies whether a role is allowed to log in to a database. A role having the **LOGIN** attribute can be considered as a user.

  Value range: If not specified, **NOLOGIN** is the default.

- **REPLICATION | NOREPLICATION**

  Specifies whether a role is allowed to initiate streaming replication or put the system in and out of backup mode. A role having the **REPLICATION** attribute is specific to replication.

  If not specified, **NOREPLICATION** is the default.

- **INDEPENDENT | NOINDEPENDENT**

  Defines private, independent roles. For a role with the INDEPENDENT attribute, administrators' permissions to control and access this role are separated. The rules are as follows:

  – Administrators have no permission to add, delete, query, modify, copy, or authorize the corresponding table objects without the authorization from the INDEPENDENT role.

  – If permissions related to private user tables are granted to non-private users, the system administrator will obtain the same permissions.

  – System administrators and security administrators with the CREATEROLE attribute have no permission to modify the inheritance relationship of the INDEPENDENT role without the authorization of the INDEPENDENT role.

  – System administrators have no permission to modify the owner of the table objects for the INDEPENDENT role.

  - System administrators and security administrators with the CREATEROLE attribute have no permission to remove the INDEPENDENT attribute of the INDEPENDENT role.
  - System administrators and security administrators with the CREATEROLE attribute have no permission to change the database password of the INDEPENDENT role. The INDEPENDENT role must manage its own password. If the password is lost, it cannot be reset.
  - The SYSADMIN attribute of a user cannot be changed to the INDEPENDENT attribute.

- **VCADMIN | NOVCADMIN**

  This parameter has no actual meaning.

- **PERSISTENCE | NOPERSISTENCE**

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

- **CONNECTION LIMIT**

  Specifies how many concurrent connections the role can make.

  NOTICE

  - The system administrator is not restricted by this parameter.
  - The number of concurrent connections of each primary database node is calculated separately (which is the value of **connlimit**). The number of all connections of the database = Value of **connlimit** x Number of normal primary database nodes.

  Value range: an integer greater than or equal to -1. The default value is **-1**, which means unlimited.

- **VALID BEGIN**

  Sets a date and time when the role's password takes effect. If this clause is omitted, the password takes effect immediately.

- **VALID UNTIL**

  Sets a date and time after which the role's password is no longer valid. If this clause is omitted, the password will be valid for all time.

- **RESOURCE POOL**

  Sets the name of resource pool used by the role. The name belongs to the system catalog **pg_resource_pool**.

- **USER GROUP**

  Creates a sub-user. This function is not supported in the current version.

- **PERM SPACE**

  Sets the space available for a user.

- **TEMP SPACE**

  Sets the space allocated to the temporary table of a user.

- **SPILL SPACE**

  Sets the operator disk flushing space of a user.

- **NODE GROUP**

  Specifies the name of the logical cluster associated with a user. (The current feature is a lab feature. Contact Huawei technical support before using it.) This function is not supported in the current version.

- **IN ROLE**

  Lists one or more existing roles to which the new role will be immediately added as a new member. It is not recommended.

- **IN GROUP**

  Specifies an obsolete spelling of **IN ROLE**. It is not recommended.

- **ROLE**

  Lists one or more existing roles which are automatically added as members of the new role.

- **ADMIN**

  Similar to ROLE. However, ADMIN grants permissions of new roles to other roles.

- **USER**

  Specifies an obsolete spelling of the ROLE clause.

- **SYSID**

  The SYSID clause is ignored.

- **DEFAULT TABLESPACE**

  The DEFAULT TABLESPACE clause is ignored.

- **PROFILE**

  The PROFILE clause is ignored.

- **PGUSER**

  In the current version, this attribute is reserved only for forward compatibility.

Examples

```
-- Create role manager whose password is xxxxxxxxx.
openGauss=# CREATE ROLE manager IDENTIFIED BY 'xxxxxxxxx';

-- Create a role with its validity from January 1, 2015 to January 1, 2026.
openGauss=# CREATE ROLE miriam WITH LOGIN PASSWORD 'xxxxxxxxx' VALID BEGIN '2015-01-01' VALID
UNTIL '2026-01-01';

-- Change the password of role manager to abcd@123.
openGauss=# ALTER ROLE manager IDENTIFIED BY 'abcd@123' REPLACE 'xxxxxxxxx';

-- Change role manager to the system administrator.
openGauss=# ALTER ROLE manager SYSADMIN;

-- Delete role manager.
openGauss=# DROP ROLE manager;

-- Delete role miriam.
openGauss=# DROP ROLE miriam;
```

Helpful Links

SET ROLE, **ALTER ROLE**, **DROP ROLE** and **GRANT**.

11.14.79 CREATE ROW LEVEL SECURITY POLICY

Function

CREATE ROW LEVEL SECURITY POLICY creates a row-level access control policy for a table.

The policy takes effect only after row-level access control is enabled (by running **ALTER TABLE... ENABLE ROW LEVEL SECURITY**). Otherwise, this statement does not take effect.

Currently, row-level access control affects the read (**SELECT**, **UPDATE**, **DELETE**) of data tables and does not affect the write (**INSERT** and **MERGE INTO**) of data tables. The table owner or system administrators can create an expression in the **USING** clause. When the client reads the data table, the database server combines the expressions that meet the condition and applies it to the execution plan in the statement rewriting phase of a query. For each tuple in a data table, if the expression returns **TRUE**, the tuple is visible to the current user; if the expression returns **FALSE** or **NULL**, the tuple is invisible to the current user.

A row-level access control policy name is specific to a table. A data table cannot have row-level access control policies with the same name. Different data tables can have the same row-level access control policy.

Row-level access control policies can be applied to specified operations (**SELECT**, **UPDATE**, **DELETE**, and **ALL**). **ALL** indicates that **SELECT**, **UPDATE**, and **DELETE** will be affected. For a new row-level access control policy, the default value **ALL** will be used if you do not specify the operations that will be affected.

Row-level access control policies can be applied to a specified user (role) or to all users (**PUBLIC**). For a new row-level access control policy, the default value **PUBLIC** will be used if you do not specify the user that will be affected.

Precautions

- Row-level access control policies can be defined for row-store tables, row-store partitioned tables, column-store tables, column-store partitioned tables, unlogged tables, and hash tables.
- Row-level access control policies cannot be defined for foreign tables and local temporary tables.
- Row-level access control policies cannot be defined for views.
- A maximum of 100 row-level access control policies can be defined for a table.
- System administrators are not affected by row-level access control policies and can view all data in a table.
- Tables queried by using SQL statements, views, functions, and stored procedures are affected by row-level access control policies.

Syntax

```
CREATE [ ROW LEVEL SECURITY ] POLICY policy_name ON table_name
    [ AS { PERMISSIVE | RESTRICTIVE } ]
    [ FOR { ALL | SELECT | UPDATE | DELETE } ]
    [ TO { role_name | PUBLIC | CURRENT_USER | SESSION_USER } [, ...] ]
    USING ( using_expression )
```

Parameter Description

- **policy_name**

  Specifies the name of a row-level access control policy to be created. The names of row-level access control policies for a table must be unique.

- **table_name**

  Specifies the name of a table to which a row-level access control policy is applied.

- **PERMISSIVE | RESTRICTIVE**

  PERMISSIVE enables the permissive policy for row-level access control. The conditions of the permissive policy are joined through the OR expression.

  RESTRICTIVE enables the restrictive policy for row-level access control. The conditions of the restrictive policy are joined through the AND expression. The join methods are as follows:

  ```
  (using_expression_permissive_1 OR using_expression_permissive_2 ...) AND
  (using_expression_restrictive_1 AND using_expression_restrictive_2 ...)
  ```

  The default value is **PERMISSIVE**.

- **command**

  Specifies the SQL operations affected by a row-level access control policy, including **ALL**, **SELECT**, **UPDATE**, and **DELETE**. If this parameter is not specified, the default value **ALL** will be used, covering **SELECT**, **UPDATE**, and **DELETE**.

  If *command* is set to **SELECT**, only tuple data that meets the condition (the return value of *using_expression* is **TRUE**) can be queried. The operations that are affected include **SELECT**, **UPDATE.... RETURNING**, and **DELETE... RETURNING**.

  If *command* is set to **UPDATE**, only tuple data that meets the condition (the return value of *using_expression* is **TRUE**) can be updated. The operations that are affected include **UPDATE**, **UPDATE ... RETURNING**, and **SELECT ... FOR UPDATE/SHARE**.

  If *command* is set to **DELETE**, only tuple data that meets the condition (the return value of *using_expression* is **TRUE**) can be deleted. The operations that are affected include **DELETE** and **DELETE ... RETURNING**.

  The following table describes the relationship between row-level access control policies and SQL statements.

Table 11-118 Relationship between row-level access control policies and SQL statements

Command	SELECT/ALL policy	UPDATE/ALL policy	DELETE/ALL policy
SELECT	Existing row	No	No
SELECT FOR UPDATE/SHARE	Existing row	Existing row	No
UPDATE	No	Existing row	No

Command	SELECT/ALL policy	UPDATE/ALL policy	DELETE/ALL policy
UPDATE RETURNING	Existing row	Existing row	No
DELETE	No	No	Existing row
DELETE RETURNING	Existing row	No	Existing row

- **role_name**

  Specifies database users affected by a row-level access control policy.

  If this parameter is not specified, the default value **PUBLIC** will be used, indicating that all database users will be affected. You can specify multiple affected database users.

  NOTICE

  System administrators are not affected by row access control.

- **using_expression**

  Specifies an expression defined for a row-level access control policy (return type: boolean).

  The expression cannot contain aggregate functions or window functions. In the statement rewriting phase of a query, if row-level access control for a data table is enabled, the expressions that meet the specified conditions will be added to the plan tree. The expression is calculated for each tuple in the data table. For **SELECT**, **UPDATE**, and **DELETE**, row data is visible to the current user only when the return value of the expression is **TRUE**. If the expression returns **FALSE**, the tuple is invisible to the current user. In this case, the user cannot view the tuple through the **SELECT** statement, update the tuple through the **UPDATE** statement, or delete the tuple through the **DELETE** statement.

Examples

```
-- Create user alice.
openGauss=# CREATE USER alice PASSWORD 'xxxxxxxxx';

-- Create user bob.
openGauss=# CREATE USER bob PASSWORD 'xxxxxxxxx';

-- Create the data table all_data.
openGauss=# CREATE TABLE public.all_data(id int, role varchar(100), data varchar(100));

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

-- Grant the read permission on the all_data table to users alice and bob.
openGauss=# GRANT SELECT ON all_data TO alice, bob;

-- Enable row-level access control.
```

```
openGauss=# ALTER TABLE all_data ENABLE ROW LEVEL SECURITY;

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

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

-- Run SELECT.
openGauss=# SELECT * FROM all_data;
 id | role  |    data
----+-------+------------
  1 | alice | alice data
  2 | bob   | bob data
  3 | peter | peter data
(3 rows)

openGauss=# EXPLAIN(COSTS OFF) SELECT * FROM all_data;
      QUERY PLAN
----------------------
 Seq Scan on all_data
(1 row)

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

openGauss=# EXPLAIN(COSTS OFF) SELECT * FROM all_data;
 QUERY PLAN
----------------------------------------------------------------
 Seq Scan on all_data
   Filter: ((role)::name = 'alice'::name)
 Notice: This query is influenced by row level security feature
(3 rows)
```

Helpful Links

DROP ROW LEVEL SECURITY POLICY, **ALTER ROW LEVEL SECURITY POLICY**

11.14.80 CREATE RULE

Function

CREATE RULE defines a new rewriting rule.

Precautions

- To define or modify rules for a table, you must be the owner of the table.

- If multiple rules of the same type are defined for the same table, the rules are triggered one by one by name in alphabetical order.
- In the view, the **RETURNING** clause can be added to the **INSERT**, **UPDATE**, and **DELETE** rules to return columns by view. If a rule is triggered by the **INSERT RETURNING**, **UPDATE RETURNING**, or **DELETE RETURNING** command, these clauses are used to calculate the output result. If a rule is triggered by a command without **RETURNING**, the **RETURNING** clause of the rule is ignored. Currently, only unconditional **INSTEAD** rules can contain the **RETURNING** clause, and only one **RETURNING** clause can exist in all rules of one event. This ensures that only one **RETURNING** clause can be used for result calculation. If the **RETURNING** clause does not exist in any valid rule, the **RETURNING** query in this view will be rejected.
- Currently, **ON SELECT** rules must be unconditional **INSTEAD** rules and must have actions consisting of a single **SELECT** command. Therefore, an **ON SELECT** rule actually turns a table into a view whose visible content is the content returned by the **SELECT** command of the rule, rather than the content in the table (if any).
- You are not advised to use column-store tables in rules, especially for write operations. The architecture implementation and transaction processing of column-store tables are greatly different from those of row-store tables. Therefore, the rule performance of column-store tables is different from that of row-store tables.

Syntax

```
CREATE [ OR REPLACE ] RULE name AS ON event
    TO table_name [ WHERE condition ]
    DO [ ALSO | INSTEAD ] { NOTHING | command | ( command ; command ... ) }
```

Events include:

```
SELECT
INSERT
DELETE
UPDATE
```

Parameter Description

- name

  Name of the created rule. It must be unique among all the rules for the same table.

  Value range: a string, which complies with the identifier naming convention and contains a maximum of 63 characters.
- event

  One of the **SELECT**, **INSERT**, **UPDATE**, and **DELETE** events.
- table_name

  Name (optionally schema-qualified) of the table or view to which the rule applies.
- condition

  SQL condition expression that returns a Boolean value, which determines whether to execute the rule. Expressions cannot reference any table except **NEW** and **OLD**, and cannot have aggregate functions. You are not advised to use numeric types such as int for **condition**, because such types can be implicitly converted to bool values (non-zero values are implicitly converted to **true** and **0** is implicitly converted to **false**), which may cause unexpected results.

- INSTEAD

  INSTEAD indicates that the initial event is replaced with this command.
- ALSO

  ALSO indicates that the command should be executed after the initial event. If neither **ALSO** nor **INSTEAD** is specified, **ALSO** is the default value.
- command

  Command that composes the rule action. A valid command is one of the **SELECT**, **INSERT**, **UPDATE**, and **DELETE** statements.

Examples

```
CREATE RULE "_RETURN" AS
   ON SELECT TO t1
   DO INSTEAD
       SELECT * FROM t2;
```

11.14.81 CREATE SCHEMA

Function

CREATE SCHEMA creates a schema.

Named objects are accessed either by "qualifying" their names with the schema name as a prefix, or by setting a search path that includes the desired schema. When creating named objects, you can also use the schema name as a prefix.

Optionally, **CREATE SCHEMA** can include sub-commands to create objects within the new schema. The sub-commands are treated essentially the same as separate commands issued after creating the schema. If the **AUTHORIZATION** clause is used, all the created objects are owned by this user.

Precautions

- Only a user with the **CREATE** permission on the current database can perform this operation.
- The owner of an object created by a system administrator in a schema with the same name as a common user is the common user, not the system administrator.

Syntax

- Create a schema based on a specified name.
```
CREATE SCHEMA schema_name
    [ AUTHORIZATION user_name ] [WITH BLOCKCHAIN] [ schema_element [ ... ] ];
```
- Create a schema based on a username.
```
CREATE SCHEMA AUTHORIZATION user_name [ schema_element [ ... ] ];
```

Parameter Description

- **schema_name**

  Specifies the schema name.

NOTICE

The name must be unique.

The schema name cannot start with **pg_**.

Value range: a string. It must comply with the naming convention rule.

- **AUTHORIZATION user_name**

  Specifies the owner of a schema. If **schema_name** is not specified, **user_name** will be used as the schema name. In this case, **user_name** can only be a role name.

  Value range: an existing username or role name

- **WITH BLOCKCHAIN**

  Specifies the tamper-proof attribute of a schema. In this mode, a row-store common user table is automatically extended to tamper-proof user table.

- **schema_element**

  Specifies an SQL statement defining an object to be created within the schema. Currently, only the **CREATE TABLE**, **CREATE VIEW**, **CREATE INDEX**, **CREATE PARTITION**, **CREATE SEQUENCE**, **CREATE TRIGGER** and **GRANT** clauses are supported.

  Objects created by sub-commands are owned by the user specified by **AUTHORIZATION**.

NOTE

If objects in the schema on the current search path are with the same name, specify the schemas for different objects. You can run **SHOW SEARCH_PATH** to check the schemas on the current search path.

Examples

```
-- Create the role1 role.
openGauss=# CREATE ROLE role1 IDENTIFIED BY 'xxxxxxxxx';

-- Create a schema named role1 for the role1 role. The owner of the films and winners tables created by
the clause is role1.
openGauss=# CREATE SCHEMA AUTHORIZATION role1
     CREATE TABLE films (title text, release date, awards text[])
     CREATE VIEW winners AS
     SELECT title, release FROM films WHERE awards IS NOT NULL;

-- Delete the schema.
openGauss=# DROP SCHEMA role1 CASCADE;
-- Delete the user.
openGauss=# DROP USER role1 CASCADE;
```

Helpful Links

ALTER SCHEMA and **DROP SCHEMA**

11.14.82 CREATE SEQUENCE

Function

CREATE SEQUENCE adds a sequence to the current database. The owner of a sequence is the user who creates the sequence.

Precautions

- A sequence is a special table that stores arithmetic progressions. It has no actual meaning and is usually used to generate unique identifiers for rows or tables.
- If a schema name is given, the sequence is created in the specified schema; otherwise, it is created in the current schema. The sequence name must be different from the names of other sequences, tables, indexes, views in the same schema.
- After the sequence is created, functions **nextval()** and **generate_series(1,N)** insert data to the table. Make sure that the number of times for invoking **nextval** is greater than or equal to N+1. Otherwise, errors will be reported because the number of times for invoking function **generate_series()** is N+1.
- By default, the maximum value of **Sequence** is 2^63 – 1. If a large identifier is used, the maximum value can be 2^127 – 1.
- A user granted with the **CREATE ANY SEQUENCE** permission can create sequences in the public and user schemas.

Syntax

```
CREATE [ LARGE ] SEQUENCE name [ INCREMENT [ BY ] increment ]
    [ MINVALUE minvalue | NO MINVALUE | NOMINVALUE ] [ MAXVALUE maxvalue | NO MAXVALUE |
NOMAXVALUE]
    [ START [ WITH ] start ] [ CACHE cache ] [ [ NO ] CYCLE | NOCYCLE ]
    [ OWNED BY { table_name.column_name | NONE } ];
```

Parameter Description

- **name**

  Specifies the name of a sequence to be created.

  Value range: a sting containing only lowercase letters, uppercase letters, special characters #_$, and digits.

- **increment**

  Specifies the step for a sequence. A positive number generates an ascending sequence, and a negative number generates a decreasing sequence.

  The default value is **1**.

- **MINVALUE minvalue | NO MINVALUE| NOMINVALUE**

  Specifies the minimum value of the sequence. If **MINVALUE** is not declared, or **NO MINVALUE** is declared, the default value of the ascending sequence is **1**, and that of the descending sequence is **$-2^{63}-1$**. **NOMINVALUE** is equivalent to **NO MINVALUE**.

- **MAXVALUE maxvalue | NO MAXVALUE| NOMAXVALUE**

  Specifies the maximum value of the sequence. If **MAXVALUE** is not declared, or **NO MAXVALUE** is declared, the default value of the ascending sequence is

2^{63}-1, and that of the descending sequence is **-1**. **NOMAXVALUE** is equivalent to **NO MAXVALUE**.

- **start**

  Specifies the start value of the sequence. The default value for an ascending sequence is **minvalue** and that for a descending sequence is **maxvalue**.

- **cache**

  Specifies the number of sequences stored in the memory for quick access purposes.

  Default value **1** indicates that one sequence can be generated each time.

  NOTE

  It is not recommended that you define **cache** and **maxvalue** or **minvalue** at the same time. The continuity of sequences cannot be ensured after **cache** is defined because unacknowledged sequences may be generated, causing waste of sequences.

- **CYCLE**

  Recycles sequences after the number of sequences reaches **maxvalue** or **minvalue**.

  If **NO CYCLE** is specified, any invocation of **nextval** would return an error after the number of sequences reaches **maxvalue** or **minvalue**.

  NOCYCLE is equivalent to **NO CYCLE**.

  The default value is **NO CYCLE**.

  If **CYCLE** is specified, the sequence uniqueness cannot be ensured.

- **OWNED BY**

  Associates a sequence with a specified column included in a table. In this way, the sequence will be deleted when you delete its associated column or the table where the column belongs to. The associated table and sequence must be owned by the same user and in the same schema. **OWNED BY** only establishes the association between a table column and the sequence. Sequences on the column do not increase automatically when data is inserted.

  The default value **OWNED BY NONE** indicates that such association does not exist.

NOTICE

You are not advised to use the sequence created using **OWNED BY** in other tables. If multiple tables need to share a sequence, the sequence must not belong to a specific table.

Examples

Create an ascending sequence named **serial**, which starts from 101.

```
openGauss=# CREATE SEQUENCE serial
 START 101
 CACHE 20;
```

Select the next number from the sequence.

```
openGauss=# SELECT nextval('serial');
 nextval
---------
     101
```

Select the next number from the sequence.

```
openGauss=# SELECT nextval('serial');
 nextval
---------
     102
```

Create a sequence associated with the table.

```
openGauss=# CREATE TABLE customer_address
(
    ca_address_sk             integer               not null,
    ca_address_id             char(16)              not null,
    ca_street_number          char(10)                      ,
    ca_street_name            varchar(60)                   ,
    ca_street_type            char(15)                      ,
    ca_suite_number           char(10)                      ,
    ca_city                   varchar(60)                   ,
    ca_county                 varchar(30)                   ,
    ca_state                  char(2)                       ,
    ca_zip                    char(10)                      ,
    ca_country                varchar(20)                   ,
    ca_gmt_offset             decimal(5,2)                  ,
    ca_location_type          char(20)
);

openGauss=# CREATE SEQUENCE serial1
 START 101
 CACHE 20
OWNED BY customer_address.ca_address_sk;
-- Delete a table and sequences.
openGauss=# DROP TABLE customer_address;
openGauss=# DROP SEQUENCE serial cascade;
openGauss=# DROP SEQUENCE serial1 cascade;
```

Helpful Links

DROP SEQUENCE and **ALTER SEQUENCE**

11.14.83 CREATE SERVER

Function

CREATE SERVER defines a new foreign server. The current feature is a lab feature. Contact Huawei technical support before using it.

Precautions

When multi-layer quotation marks are used for sensitive columns (such as **password** and **secret_access_key**) in OPTIONS, the semantics is different from that in the scenario where quotation marks are not used. Therefore, sensitive columns are not identified for anonymization.

Syntax

```
CREATE SERVER server_name
    FOREIGN DATA WRAPPER fdw_name
    OPTIONS ( { option_name ' value ' } [, ...] ) ;
```

Parameter Description

- **server_name**

  Specifies the server name.

  Value range: a string containing no more than 63 characters

- **fdw_name**

  Specifies the name of the foreign data wrapper.

  Value range: **dist_fdw**, **log_fdw**, and **file_fdw**

- **OPTIONS ({ option_name ' value ' } [, ...])**

  Specifies options for the server. These options typically define the connection details of the server, but the actual names and values depend on the foreign data wrapper of the server.

  – Specifies the parameters for the foreign server. The detailed parameter description is as follows:

    ■ encrypt

      Specifies whether data is encrypted. This parameter is available only when **type** is **OBS**. The default value is **on**.

      Value range:

      ○ **on** indicates that data is encrypted and HTTPS is used for communication.

      ○ **off** indicates that data is not encrypted and HTTP is used for communication.

    ■ access_key

      Specifies the access key (AK) (obtained by users from the OBS console) used for the OBS access protocol. When you create a foreign table, the AK value is encrypted and saved to the metadata table of the database. This parameter is available only when **type** is set to **OBS**.

    ■ secret_access_key

      Specifies the secret key (SK) value (obtained by users from the OBS console) used for the OBS access protocol. When you create a foreign table, the SK value is encrypted and saved to the metadata table of the database. This parameter is available only when **type** is set to **OBS**.

In addition to the connection parameters supported by libpq, the following parameters are provided:

- **fdw_startup_cost**

  Estimates the startup time required for a foreign table scan, including the time to establish a connection, analyze the request at the remote server, and generate a plan. The default value is **100**.

- **fdw_typle_cost**

  Specifies the additional consumption when each tuple is scanned on a remote server. The value specifies the extra consumption of data transmission between servers. The default value is **0.01**.

Examples

Create a server.

```
openGauss=* create server my_server foreign data wrapper log_fdw;
CREATE SERVER
```

Helpful Links

ALTER SERVER and **DROP SERVER**

11.14.84 CREATE SUBSCRIPTION

Function

CREATE SUBSCRIPTION adds a new subscription to the current database. Only the system administrator can create a subscription. The subscription name must be different from that of any existing subscription in the database. A subscription represents a replication connection to a publisher. Therefore, this command not only adds definitions to the local system catalog, but also creates replication slots on the publication side. When the transaction that runs this command is committed, the logical replication thread is started to replicate the newly subscribed data.

Precautions

When a replication slot is created (default behavior), **CREATE SUBSCRIPTION** cannot be executed in a transaction block. Currently, a maximum of 65,534 subscriptions (including enabled and disabled subscriptions) are supported.

Syntax

```
CREATE SUBSCRIPTION subscription_name
   CONNECTION 'conninfo'
   PUBLICATION publication_name [, ...]
   [ WITH ( subscription_parameter [= value] [, ... ] ) ]
```

Parameter Description

- **subscription_name**

  Specifies the name of a new subscription.
- **CONNECTION 'conninfo'**

  Specifies the character string for connecting to the publication side.

  For example, **'host=1.1.1.1,2.2.2.2 port=10000,20000 dbname=postgres user=repusr1 password=password_123'**.

  For details about the fields in the character string, see **Link Parameters**. The following table lists the common link parameters.
  - **host**

    IP address of the publisher. You can specify the IP addresses of the primary and standby nodes of the publisher at the same time. If multiple IP addresses are specified, separate them with commas (,).
  - **port**

The port number of the publication side cannot be the primary port number. The port number must be the primary port number plus 1. Otherwise, the port number conflicts with the thread pool.

CAUTION

The number of hosts must be the same as that of ports.

  - **dbname**

    Specifies the database where a publication is located.

  - **user** and **password**

    Specify the username and password used to connect to the publication side. The user has the system administrator permission (**SYSADMIN**) or O&M administrator permission (**OPRADMIN**). The password must be encrypted. Before creating a subscription, run the **gs_guc generate -S xxxxxx -D $GAUSSHOME/bin -o subscription** command on the subscription side.

- **PUBLICATION publication_name**

  Specifies the name of the publication to be subscribed to on the publication side. A subscription can correspond to multiple publications.

- **WITH (subscription_parameter [= value] [, ...])**

  Specifies the optional parameters for a subscription. The following parameters are supported:

  - **enabled (boolean)**

    Specifies whether a subscription should be actively replicated, or whether it should be just set but not started. The default value is **true**.

  - **slot_name (string)**

    Specifies the name of the replication slot to be used. By default, the subscription name is used as the replication slot name.

    If **enabled** is set to **false** during subscription creation, **slot_name** is forcibly set to **NONE** which indicates a null value. In this case, the replication slot does not exist even if the value of **slot_name** is specified.

  - **synchronous_commit (enum)**

    The value of this parameter overwrites the value of **synchronous_commit**. The default value is **off**.

    It is safe to use the value **off** for logical replication. If the subscription side loses the transaction due to a lack of synchronization, the data is sent again from the publisher. A different setting may be appropriate for synchronous logical replication. The logical replication thread reports the locations of WRITE and REFRESH operations to the publication side. When synchronous replication is used, the publication side waits for the actual REFRESH operations. This means that setting the subscriber's **synchronous_commit** to **off** when the subscription is used for synchronous replication may increase the latency of **COMMIT** on the publication server. In this case, it is advantageous to set **synchronous_commit** to **local** or a higher value.

  - **binary (boolean)**

Specifies whether the subscription is sent by the publisher in binary format. The value **true** indicates that the data is sent in binary format, and the value **false** indicates that the data is sent in the default text format. Default value: **false**

Examples

```
-- Create a subscription to a remote server, replicate tables in the mypublication and insert_only publications, and start replication immediately upon commit.
CREATE SUBSCRIPTION mysub
         CONNECTION 'host=192.168.1.50 port=5432 user=foo dbname=foodb password=xxxx'
        PUBLICATION mypublication, insert_only;
-- Create a subscription to a remote server, replicate the tables in the insert_only publication, and do not start replication immediately until it is enabled later.
CREATE SUBSCRIPTION mysub
         CONNECTION 'host=192.168.1.50 port=5432 user=foo dbname=foodb password=xxxx '
        PUBLICATION insert_only
               WITH (enabled = false);
-- Modify the connection information of a subscription.
ALTER SUBSCRIPTION mysub CONNECTION 'host=192.168.1.51 port=5432 user=foo dbname=foodb password=xxxx';
-- Enable a subscription.
ALTER SUBSCRIPTION mysub SET(enabled=true);
-- Delete a subscription.
DROP SUBSCRIPTION mysub;
```

Helpful Links

ALTER SUBSCRIPTION and DROP SUBSCRIPTION

11.14.85 CREATE SYNONYM

Function

CREATE SYNONYM creates a synonym object. A synonym is an alias of a database object and is used to record the mapping between database object names. You can use synonyms to access associated database objects.

Precautions

- The user of a synonym should be its owner.
- If the schema name is specified, create a synonym in the specified schema. Otherwise create a synonym in the current schema.
- Database objects that can be accessed using synonyms include tables, views, functions, and stored procedures.
- To use synonyms, you must have the required permissions on associated objects.
- The following DML statements support synonyms: **SELECT**, **INSERT**, **UPDATE**, **DELETE**, **EXPLAIN**, and **CALL**.
- You are not advised to create synonyms for temporary tables. To create a synonym, you need to specify the schema name of the target temporary table. Otherwise, the synonym cannot be used normally. In addition, you need to run the **DROP SYNONYM** command before the current session ends.
- After an original object is deleted, the synonym associated with the object will not be deleted in cascading mode. If you continue to access the synonym, an error message is displayed, indicating that the synonym has expired.

- Synonyms cannot be created for encrypted tables that contain encrypted columns and views, functions, and stored procedures based on encrypted tables.

Syntax

```
CREATE [ OR REPLACE ] SYNONYM synonym_name
    FOR object_name;
```

Parameter Description

- **synonym**

  Specifies the name of the synonym to be created, which can contain the schema name.

  Value range: a string. It must comply with the identifier naming convention.

- **object_name**

  Specifies the name of an object that is associated (optionally with schema names).

  Value range: a string. It must comply with the identifier naming convention.

  NOTE

  object_name can be the name of an object that does not exist.

  CAUTION

  Do not create aliases for functions that contain passwords and other sensitive information, such as the encryption function gs_encrypt and the decryption function gs_decrypt or use aliases to call the functions to prevent sensitive information leakage.

Examples

```
-- Create schema ot.
openGauss=# CREATE SCHEMA ot;

-- Create table ot.t1 and its synonym t1.
openGauss=# CREATE TABLE ot.t1(id int, name varchar2(10));
openGauss=# CREATE OR REPLACE SYNONYM t1 FOR ot.t1;

-- Use synonym t1.
openGauss=# SELECT * FROM t1;
openGauss=# INSERT INTO t1 VALUES (1, 'ada'), (2, 'bob');
openGauss=# UPDATE t1 SET t1.name = 'cici' WHERE t1.id = 2;

-- Create synonym v1 and its associated view ot.v_t1.
openGauss=# CREATE SYNONYM v1 FOR ot.v_t1;
openGauss=# CREATE VIEW ot.v_t1 AS SELECT * FROM ot.t1;

-- Use synonym v1.
openGauss=# SELECT * FROM v1;

-- Create overloaded function ot.add and its synonym add.
openGauss=# CREATE OR REPLACE FUNCTION ot.add(a integer, b integer) RETURNS integer AS
$$
SELECT $1 + $2
$$
LANGUAGE sql;
```

```
openGauss=# CREATE OR REPLACE FUNCTION ot.add(a decimal(5,2), b decimal(5,2)) RETURNS
decimal(5,2) AS
$$
SELECT $1 + $2
$$
LANGUAGE sql;

openGauss=# CREATE OR REPLACE SYNONYM add FOR ot.add;

-- Use synonym add.
openGauss=# SELECT add(1,2);
openGauss=# SELECT add(1.2,2.3);

-- Create stored procedure ot.register and its synonym register.
openGauss=# CREATE PROCEDURE ot.register(n_id integer, n_name varchar2(10))
SECURITY INVOKER
AS
BEGIN
    INSERT INTO ot.t1 VALUES(n_id, n_name);
END;
/

openGauss=# CREATE OR REPLACE SYNONYM register FOR ot.register;

-- Use synonym register to invoke the stored procedure.
openGauss=# CALL register(3,'mia');

-- Delete the synonym.
openGauss=# DROP SYNONYM t1;
openGauss=# DROP SYNONYM IF EXISTS v1;
openGauss=# DROP SYNONYM IF EXISTS add;
openGauss=# DROP SYNONYM register;
openGauss=# DROP SCHEMA ot CASCADE;
```

Helpful Links

ALTER SYNONYM and **DROP SYNONYM**

11.14.86 CREATE TABLE

Function

CREATE TABLE creates an initially empty table in the current database. The table will be owned by the creator.

Precautions

- For details about the data types supported by column-store tables, see **Data Types Supported by Column-store Tables**.
- Column-store tables do not support the array.
- Column-store tables do not support column generation.
- Column-store tables cannot be created as global temporary tables.
- It is recommended that the number of column-store tables do not exceed 1000.
- If an error occurs during table creation, after it is fixed, the system may fail to delete the empty disk files created before the last automatic clearance. This problem seldom occurs and does not affect system running of the database.

- Only **PARTIAL CLUSTER KEY**, **UNIQUE**, and **PRIMARY KEY** can be used as the table-level constraint of column-store tables. Table-level foreign key constraints are not supported.
- Only the **NULL**, **NOT NULL**, **DEFAULT** constant values, **UNIQUE**, and **PRIMARY KEY** can be used as column-store table constraints.
- Whether column-store tables support a delta table is specified by the enable_delta_store parameter. The threshold for storing data into a delta table is specified by the **deltarow_threshold** parameter.
- When JDBC is used, the **DEFAULT** value can be set through **PrepareStatement**.
- A user granted with the CREATE ANY TABLE permission can create tables in the public and user schemas. To create a table that contains serial columns, you must also grant the CREATE ANY SEQUENCE permission to create sequences.

Syntax

Create a table.

```
CREATE [ [ GLOBAL | LOCAL ] [ TEMPORARY | TEMP ] | UNLOGGED ] TABLE [ IF NOT EXISTS ] table_name
    ({ column_name data_type [ compress_mode ] [ COLLATE collation ] [ column_constraint [ ... ] ]
        | table_constraint
        | LIKE source_table [ like_option [...] ] }
        [, ... ])
    [ WITH ( {storage_parameter = value} [, ... ] ) ]
    [ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
    [ COMPRESS | NOCOMPRESS ]
    [ TABLESPACE tablespace_name ];
```

- **column_constraint** is as follows:

```
[ CONSTRAINT constraint_name ]
{ NOT NULL |
  NULL |
  CHECK ( expression ) |
  DEFAULT default_expr |
  GENERATED ALWAYS AS ( generation_expr ) STORED |
  UNIQUE index_parameters |
  ENCRYPTED WITH ( COLUMN_ENCRYPTION_KEY = column_encryption_key, ENCRYPTION_TYPE =
encryption_type_value ) |
  PRIMARY KEY index_parameters |
  REFERENCES reftable [ ( refcolumn ) ] [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ]
      [ ON DELETE action ] [ ON UPDATE action ] }
[ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

- **compress_mode** of a column is as follows:

```
{ DELTA | PREFIX | DICTIONARY | NUMSTR | NOCOMPRESS }
```

- **table_constraint** is as follows:

```
[ CONSTRAINT constraint_name ]
{ CHECK ( expression ) |
  UNIQUE ( column_name [, ... ] ) index_parameters |
  PRIMARY KEY ( column_name [, ... ] ) index_parameters |
  FOREIGN KEY ( column_name [, ... ] ) REFERENCES reftable [ (refcolumn [, ... ] ) ]
      [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ] [ ON DELETE action ] [ ON UPDATE action ]
|
  PARTIAL CLUSTER KEY ( column_name [, ... ] ) }
[ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

- **like_option** is as follows:

```
{ INCLUDING | EXCLUDING } { DEFAULTS | GENERATED | CONSTRAINTS | INDEXES | STORAGE |
COMMENTS | PARTITION | RELOPTIONS | ALL }
```

- **index_parameters** is as follows:

```
[ WITH ( {storage_parameter = value} [, ... ] ) ]
[ USING INDEX TABLESPACE tablespace_name ]
```

Parameter Description

- **UNLOGGED**

  If this keyword is specified, the created table is an unlogged table. Data written to unlogged tables is not written to the WALs, which makes them considerably faster than ordinary tables. However, an unlogged table is automatically truncated after conflicts, operating system restart, database restart, primary/standby switchover, power-off, or abnormal shutdown, incurring data loss risks. Contents of an unlogged table are also not replicated to standby nodes. Any indexes created on an unlogged table are not automatically logged as well.

  Usage scenario: Unlogged tables do not ensure data security. Users can back up data before using unlogged tables; for example, users should back up the data before a system upgrade.

  Troubleshooting: If data is missing in the indexes of unlogged tables due to some unexpected operations such as an unclean shutdown, users should re-create the indexes with errors.

- **GLOBAL | LOCAL**

  When creating a temporary table, you can specify the **GLOBAL** or **LOCAL** keyword before **TEMP** or **TEMPORARY**. If the keyword **GLOBAL** is specified, GaussDB creates a global temporary table. Otherwise, GaussDB creates a local temporary table.

- **TEMPORARY | TEMP**

  If **TEMP** or **TEMPORARY** is specified, the created table is a temporary table. Temporary tables are classified into global temporary tables and local temporary tables. If the keyword **GLOBAL** is specified when a temporary table is created, the table is a global temporary table. Otherwise, the table is a local temporary table.

  The metadata of the global temporary table is visible to all sessions. After the sessions end, the metadata still exists. The user data, indexes, and statistics of a session are isolated from those of another session. Each session can only view and modify the data committed by itself. Global temporary tables have two schemas: **ON COMMIT PRESERVE ROWS** and **ON COMMIT PRESERVE ROWS**. In session-based **ON COMMIT PRESERVE ROWS** schema, user data is automatically cleared when a session ends. In transaction-based **ON COMMIT DELETE ROWS** schema, user data is automatically cleared when the commit or rollback operation is performed. If the **ON COMMIT** option is not specified during table creation, the session level is used by default. Different from local temporary tables, you can specify a schema that does not start with **pg_temp_** when creating a global temporary table.

  A local temporary table is automatically dropped at the end of the current session. Therefore, you can create and use temporary tables in the current session as long as the connected database node in the session is normal. Temporary tables are created only in the current session. If a DDL statement involves operations on temporary tables, a DDL error will be generated. Therefore, you are not advised to perform operations on temporary tables in DDL statements. **TEMP** is equivalent to **TEMPORARY**.

NOTICE

- Local temporary tables are visible to the current session through the schema starting with **pg_temp**. Users should not delete schemas starting with **pg_temp** or **pg_toast_temp**.
- If **TEMPORARY** or **TEMP** is not specified when you create a table but its schema is set to that starting with **pg_temp_** in the current session, the table will be created as a temporary table.
- If global temporary tables and indexes are being used by other sessions, do not perform **ALTER** or **DROP** (except the **ALTER INDEX index_name REBUILD** command).
- The DDL of a global temporary table affects only the user data and indexes of the current session. For example, **TRUNCATE**, **REINDEX**, and **ANALYZE** are valid only for the current session.
- You can set the GUC parameter **max_active_global_temporary_table** to determine whether to enable the global temporary table function. If **max_active_global_temporary_table** is set to **0**, the global temporary table function is disabled.
- A temporary table is visible only to the current session. Therefore, it cannot be used together with **\parallel on**.
- Temporary tables do not support the primary/standby switchover.
- The global temporary table does not respond to automatic clearance. In persistent connection scenarios, you are advised to use the global temporary table in the ON COMMIT DELETE ROWS clause or periodically and manually execute the VACUUM statement. Otherwise, Clogs may not be recycled.

- **IF NOT EXISTS**

  Sends a notice, but does not throw an error, if a table with the same name exists.
- **table_name**

  Specifies the name of the table to be created.

NOTICE

Some processing logic of materialized views determines whether a table is the log table of a materialized view or a table associated with a materialized view based on the table name prefix. Therefore, do not create a table whose name prefix is **mlog_** or **matviewmap_**. Otherwise, some functions of the table are affected.

- **column_name**

  Specifies the name of a column to be created in the new table.
- **data_type**

  Specifies the data type of the column.
- **compress_mode**

Specifies whether to compress a table column. The option specifies the algorithm preferentially used by table columns. Row-store tables do not support compression.

Value range: **DELTA**, **PREFIX**, **DICTIONARY**, **NUMSTR**, and **NOCOMPRESS**

- **COLLATE collation**

  Assigns a collation to the column (which must be of a collatable data type). If no collation is specified, the default collation is used. You can run the **select * from pg_collation;** command to query collation rules from the **pg_collation** system catalog. The default collation rule is the row starting with **default** in the query result.

- **LIKE source_table [like_option ...]**

  Specifies a table from which the new table automatically copies all column names, their data types, and their not-null constraints.

  The new table and the original table are decoupled after creation is complete. Changes to the original table will not be applied to the new table, and it is not possible to include data of the new table in scans of the original table.

  Columns and constraints copied by **LIKE** are not merged with the same name. If the same name is specified explicitly or in another **LIKE** clause, an error is reported.

  – The default expressions are copied from the original table to the new table only if **INCLUDING DEFAULTS** is specified. The default behavior is to exclude default expressions, resulting in the copied columns in the new table having default values **NULL**.

  – If **INCLUDING GENERATED** is specified, the generated expression of the source table column is copied to the new table. By default, the generated expression is not copied.

  – The **CHECK** constraints are copied from the original table to the new table only when **INCLUDING CONSTRAINTS** is specified. Other types of constraints are never copied to the new table. Not-null constraints are always copied to the new table. These rules also apply to column constraints and table constraints.

  – Any indexes on the original table will not be created on the new table, unless the **INCLUDING INDEXES** clause is specified.

  – **STORAGE** settings for the copied column definitions are copied only if **INCLUDING STORAGE** is specified. The default behavior is to exclude **STORAGE** settings.

  – If **INCLUDING COMMENTS** is specified, comments for the copied columns, constraints, and indexes are copied. The default behavior is to exclude comments.

  – If **INCLUDING PARTITION** is specified, the partition definitions of the source table are copied to the new table, and the new table no longer uses the **PARTITION BY** clause. The default behavior is to exclude partition definition of the original table. If the source table has an index, you can use the **INCLUDING PARTITION INCLUDING INDEXES** syntax. If only **INCLUDING INDEXES** is used for a partitioned table, the target table will be defined as an ordinary table, but the index is a partitioned index. In this case, an error will be reported because ordinary tables do not support partitioned indexes.

- If **INCLUDING RELOPTIONS** is specified, the new table will copy the storage parameter (that is, **WITH** clause) of the source table. The default behavior is to exclude partition definition of the storage parameter of the original table.
- **INCLUDING ALL** contains the meaning of **INCLUDING DEFAULTS**, **INCLUDING GENERATED**, **INCLUDING CONSTRAINTS**, **INCLUDING INDEXES**, **INCLUDING STORAGE**, **INCLUDING COMMENTS**, **INCLUDING PARTITION**, and **INCLUDING RELOPTIONS**.

NOTICE

- If the source table contains a sequence with the **SERIAL**, **BIGSERIAL**, **SMALLSERIAL** or **LARGESERIAL** data type, or a column in the source table is a sequence by default and the sequence is created for this table by using **CREATE SEQUENCE... OWNED BY**, these sequences will not be copied to the new table, and another sequence specific to the new table will be created. This is different from earlier versions. To share a sequence between the source table and new table, create a shared sequence (do not use **OWNED BY**) and set a column in the source table to this sequence.
- You are not advised to set a column in the source table to the sequence specific to another table especially when the table is distributed in specific node groups, because doing so may result in **CREATE TABLE ... LIKE** execution failures. In addition, doing so may cause the sequence to become invalid in the source sequence because the sequence will also be deleted from the source table when it is deleted from the table that the sequence is specific to. To share a sequence among multiple tables, you are advised to create a shared sequence for them.
- **EXCLUDING** of a partitioned table must be used together with **INCLUDING ALL**, for example, **INCLUDING ALL EXCLUDING DEFAULTS**, except for **DEFAULTS** of the source partitioned table.

- **WITH ({ storage_parameter = value } [, ...])**

  Specifies an optional storage parameter for a table or an index. The WITH clause for a table can contain **OIDS=TRUE** or **OIDS** to specify that each row in the new table is assigned an OID. If **OIDS=FALSE** is specified, no OID is assigned.

  NOTE

  When using **Numeric** of any precision to define a column, specifies precision **p** and scale **s**. When precision and scale are not specified, the input will be displayed.

  The description of parameters is as follows:

  - FILLFACTOR

    The fill factor of a table is a percentage from 10 to 100. **100** (complete filling) is the default value. When a smaller fill factor is specified, **INSERT** operations fill table pages only to the indicated percentage. The remaining space on each page is reserved for updating rows on that page. This gives **UPDATE** a chance to place the updated copy of a row on the same page, which is more efficient than placing it on a different page. For a table whose entries are never updated, setting the fill factor to **100** (complete filling) is the best choice, but in heavily updated tables a smaller fill factor would be appropriate. The parameter has no meaning for column-store tables.

Value range: 10–100

– ORIENTATION

Specifies the storage mode (row-store or column-store) of table data. This parameter cannot be modified once it is set.

Value range:

- **ROW** indicates that table data is stored in rows.

  ROW applies to OLTP service and scenarios with a large number of point queries or addition/deletion operations.

- **COLUMN** indicates that the data is stored in columns.

  COLUMN applies to the data warehouse service, which has a large amount of aggregation computing, and involves a few column operations.

Default value:

If an ordinary tablespace is specified, the default is **ROW**.

– STORAGE_TYPE

Specifies the storage engine type. This parameter cannot be modified once it is set.

Value range:

- **USTORE** indicates that tables support the inplace-update storage engine. Note that the **track_counts** and **track_activities** parameters must be enabled when the Ustore table is used. Otherwise, space expansion may occur.

  CAUTION

  Currently, the Ustore storage engine does not support the ultimate RTO playback mode. For a primary node, if **recovery_parse_workers** is set to a value greater than 1, an error is reported when Ustore tables are created. For a standby node, if the database contains Ustore tables, playback may fail and an error may be reported if the ultimate RTO function is enabled. In serious cases, data on the standby node may be damaged. In this case, you need to rebuild the standby node.

- **ASTORE** indicates that tables support the append-only storage engine.

Default value:

If no table is specified, data is stored in append-only mode by default.

– INIT_TD

Specifies the number of TDs to be initialized when an Ustore table is created. This parameter is valid only when an Ustore table is created.

Value ranges: 2–128. The default value is **4**.

- COMPRESSION

  Specifies the compression level of table data. It determines the compression ratio and time. Generally, the higher the level of compression, the higher the ratio, the longer the time; and the lower the level of compression, the lower the ratio, the shorter the time. The actual compression ratio depends on the distribution mode of table data loaded. Row-store tables do not support compression.

  Value range:

  The valid values for column-store tables are **YES**, **NO**, **LOW**, **MIDDLE**, and **HIGH**, and the default value is **LOW**.

- COMPRESSLEVEL

  Specifies the table data compression ratio and duration at the same compression level. This divides a compression level into sublevels, providing more choices for compression ratio and duration. As the value becomes greater, the compression ratio becomes higher and duration longer at the same compression level.

  Value range: 0 to 3. The default value is **0**.

- MAX_BATCHROW

  Specifies the maximum number of records in a storage unit during data loading. The parameter is only valid for column-store tables.

  Value range: 10000 to 60000. The default value is **60000**.

- PARTIAL_CLUSTER_ROWS

  Specifies the number of records to be partially clustered for storage during data loading. The parameter is only valid for column-store tables.

  Value range: greater than or equal to **MAX_BATCHROW**. You are advised to set this parameter to an integer multiple of **MAX_BATCHROW**.

- DELTAROW_THRESHOLD

  Specifies the upper limit of to-be-imported rows for triggering the data import to a delta table when data of a column-store table is to be imported. This parameter takes effect only if **enable_delta_store** is set to **on**. The parameter is only valid for column-store tables.

  Value range: 0 to 9999. The default value is **100**.

- segment

  The data is stored in segment-page mode. This parameter supports only row-store tables. Column-store tables, temporary tables, and unlogged tables are not supported. The Ustore storage engine is not supported.

  Value range: **on** and **off**

  Default value: **off**

- enable_tde

  Creates a transparent encryption table. Before enabling this function, ensure that the **enable_tde** in "GUC Parameter" has been enabled, the KMS service has been enabled, and the database instance master key ID in **tde_cmk_id** in "GUC Parameter" has been correctly configured. This parameter supports only row-store tables. Column-store tables and temporary tables are not supported. The Ustore storage engine is not supported.

Value range: **on** and **off** If this parameter is set to **on**, transparent data encryption is enabled. If this parameter is set to **off**, transparent data encryption is disabled but the encryption function will be enabled later. When a table is created, a data encryption key will be applied from KMS.

Default value: **off**

– parallel_workers

Number of bgworker threads started when an index is created. For example, value **2** indicates that two bgworker threads are started to create indexes concurrently.

Value range: [0,32], int type. The value **0** indicates that this function is disabled.

Default value: If this parameter is not set, the concurrent index creation function is disabled.

– encrypt_algo

Specifies the transparent data encryption algorithm. Before enabling this function, ensure that **enable_tde** must be set for a table. The encryption algorithm can be specified only when a table is created. Different tables support different encryption algorithms. After the table is created, the encryption algorithm cannot be changed.

Value range: a string. The value can be **AES_128_CTR** or **SM4_CTR**.

If **enable_tde** is not set, the default value is null. If **enable_tde** is set to **on** or **off** and **encrypt_algo** is not set, the value is **AES_128_CTR**.

– dek_cipher

Ciphertext of the key used for transparent data encryption. When **enable_tde** is enabled, the system automatically applies for ciphertext creation. You cannot specify the ciphertext. The key rotation function can be used to update the key.

Value range: a string.

If encryption is disabled, the default value is null by default.

– cmk_id

Master key ID of the database instance used for transparent data encryption. When **enable_tde** is enabled, the value is obtained from **tde_cmk_id** in "GUC Parameter" and cannot be specified or modified by users.

Value range: a string.

If encryption is disabled, the default value is null by default.

– hasuids

If this parameter is set to **on**, a unique table-level ID is allocated to a tuple when the tuple is updated.

Value range: **on** and **off**

Default value: **off**

– min_tuples

The optimizer selects the larger value of the estimated statistics and the parameter to calculate the data volume based on the estimation table of statistics.

Value range: [0,DBL_MAX)

Default value: **0**

- **WITHOUT OIDS**

  It is equivalent to **WITH(OIDS=FALSE)**.

- **ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP }**

  ON COMMIT determines what to do when you commit a temporary table creation operation. The three options are as follows. Currently, only **PRESERVE ROWS** and **DELETE ROWS** can be used.

  - **PRESERVE ROWS** (default): No special action is taken at the ends of transactions. The temporary table and its table data are unchanged.
  - **DELETE ROWS**: All rows in the temporary table will be deleted at the end of each transaction block.
  - **DROP**: The temporary table will be dropped at the end of the current transaction block. Only local temporary tables can be dropped. Global temporary tables cannot be dropped.

- **COMPRESS | NOCOMPRESS**

  If you specify **COMPRESS** in the **CREATE TABLE** statement, the compression feature is triggered in case of a bulk **INSERT** operation. If this feature is enabled, a scan is performed for all tuple data within the page to generate a dictionary and then the tuple data is compressed and stored. If **NOCOMPRESS** is specified, the table is not compressed. Row-store tables do not support compression.

  Default value: **NOCOMPRESS**, that is, tuple data is not compressed before storage.

- **TABLESPACE tablespace_name**

  Specifies the tablespace where the new table is created. If not specified, the default tablespace is used.

- **CONSTRAINT constraint_name**

  Specifies the name of a column or table constraint. The optional constraint clauses specify constraints that new or updated rows must satisfy for an INSERT or UPDATE operation to succeed.

  There are two ways to define constraints:

  - A column constraint is defined as part of a column definition, and it is bound to a particular column.
  - A table constraint is not bound to a particular column but can apply to more than one column.

- **NOT NULL**

  The column is not allowed to contain null values.

- **NULL**

  The column is allowed to contain null values. This is the default setting.

  This clause is only provided for compatibility with non-standard SQL databases. It is not recommended.

- **CHECK (expression)**

  Specifies an expression producing a Boolean result where the INSERT or UPDATE operation of new or updated rows can succeed only when the expression result is **TRUE** or **UNKNOWN**; otherwise, an error is thrown and the database is not altered.

A CHECK constraint specified as a column constraint should reference only the column's value, while an expression in a table constraint can reference multiple columns.

NOTE

<>NULL and **!=NULL** are invalid in an expression. Change them to **IS NOT NULL**.

- **DEFAULT default_expr**

  Assigns a default data value to a column. The value can be any variable-free expressions. (Subqueries and cross-references to other columns in the current table are not allowed.) The data type of the default expression must match the data type of the column.

  The default expression will be used in any insert operation that does not specify a value for the column. If there is no default value for a column, then the default value is null.

- **GENERATED ALWAYS AS (generation_expr) STORED**

  This clause creates a column as a generated column. The value of the generated column is calculated by **generation_expr** when data is written (inserted or updated). **STORED** indicates that the value of the generated column is stored as a common column.

  NOTE

  - The generation expression cannot refer to data other than the current row in any way. The generation expression cannot reference other generation columns or system columns. The generation expression cannot return a result set. No subquery, aggregate function, or window function can be used. The function called by the generation expression can only be an immutable function.
  - Default values cannot be specified for generated columns.
  - The generated column cannot be used as a part of the partition key.
  - Do not specify the generated column and the CASCADE, SET NULL, and SET DEFAULT actions of the ON UPDATE constraint at the same time. Do not specify the generated column and the SET NULL, and SET DEFAULT actions of the ON DELETE constraint at the same time.
  - The method of modifying and deleting generated columns is the same as that of common columns. Delete the common column that the generated column depends on. The generated column is automatically deleted. The type of the column on which the generated column depends cannot be changed.
  - The generated column cannot be directly written. In the INSERT or UPDATE statement, values cannot be specified for generated columns, but the keyword DEFAULT can be specified.
  - The permission control for generated columns is the same as that for common columns.
  - Columns cannot be generated for column-store tables and memory-optimized tables (MOTs). In foreign tables, only **postgres_fdw** supports generated columns.

- **UNIQUE index_parameters**

  UNIQUE (column_name [, ...]) index_parameters

  Specifies that a group of one or more columns of a table can contain only unique values.

  For the purpose of a unique constraint, null is not considered equal.

- **PRIMARY KEY index_parameters**

  PRIMARY KEY (column_name [, ...]) index_parameters

Specifies that a column or columns of a table can contain only unique (non-duplicate) and non-null values.

Only one primary key can be specified for a table.

- **REFERENCES reftable [(refcolum)] [MATCH matchtype] [ON DELETE action] [ON UPDATE action] (column constraint)**

  FOREIGN KEY (column_name [, ...]) REFERENCES reftable [(refcolumn [, ...])] [MATCH matchtype] [ON DELETE action] [ON UPDATE action] (table constraint)

  The foreign key constraint requires that the group consisting of one or more columns in the new table should contain and match only the referenced column values in the referenced table. If **refcolum** is omitted, the primary key of **reftable** is used. The referenced column should be the only column or primary key in the referenced table. A foreign key constraint cannot be defined between a temporary table and a permanent table.

  There are three types of matching between a reference column and a referenced column:

  – **MATCH FULL**: A column with multiple foreign keys cannot be **NULL** unless all foreign key columns are **NULL**.

  – **MATCH SIMPLE** (default): Any unexpected foreign key column can be **NULL**.

  – **MATCH PARTIAL**: This option is not supported currently.

  In addition, when certain operations are performed on the data in the referenced table, the operations are performed on the corresponding columns in the new table. **ON DELETE**: specifies the operations to be executed after a referenced row in the referenced table is deleted. **ON UPDATE**: specifies the operation to be performed when the referenced column data in the referenced table is updated. Possible responses to the **ON DELETE** and **ON UPDATE** clauses are as follows:

  – **NO ACTION** (default): An error indicating that the foreign key constraint is violated is reported. If the constraint is deferrable and there are still any referenced rows, this error will occur when the constraint is checked.

  – **RESTRICT**: An error indicating that the foreign key constraint is violated is created. It is the same as **NO ACTION** except that the constraint is not deferrable.

  – **CASCADE**: deletes any rows referencing the deleted row, or update the value of the referencing column to the new value of the referenced column, respectively.

  – **SET NULL**: sets the referencing column(s) to **NULL**.

  – **SET DEFAULT**: sets the referenced columns to their default values.

- **DEFERRABLE | NOT DEFERRABLE**

  Controls whether the constraint can be deferred. A constraint that is not deferrable will be checked immediately after every command. Checking of constraints that are deferrable can be postponed until the end of the transaction using the **SET CONSTRAINTS** command. **NOT DEFERRABLE** is the default value. Currently, only UNIQUE constraints, primary key constraints, and foreign key constraints accept this clause. All the other constraints are not deferrable.

NOTE

Ustore tables do not support the keywords **DEFERRABLE** and **INITIALLY DEFERRED**.

- **PARTIAL CLUSTER KEY**

  Specifies a partial cluster key for storage. When importing data to a column-store table, you can perform local data sorting by specified columns (single or multiple).

- **INITIALLY IMMEDIATE | INITIALLY DEFERRED**

  If a constraint is deferrable, this clause specifies the default time to check the constraint.

  - If the constraint is **INITIALLY IMMEDIATE** (default value), it is checked after each statement.
  - If the constraint is **INITIALLY DEFERRED**, it is checked only at the end of the transaction.

  The constraint check time can be altered using the **SET CONSTRAINTS** statement.

- **USING INDEX TABLESPACE tablespace_name**

  Allows selection of the tablespace in which the index associated with a **UNIQUE** or **PRIMARY KEY** constraint will be created. If not specified, **default_tablespace** is consulted, or the default tablespace in the database if **default_tablespace** is empty.

- **ENCRYPTION_TYPE = encryption_type_value**

  For the encryption type in the ENCRYPTED WITH constraint, the value of **encryption_type_value** is **DETERMINISTIC** or **RANDOMIZED**.

Examples

```
-- Create a simple table.
openGauss=# CREATE TABLE tpcds.warehouse_t1
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)                   ,
    W_WAREHOUSE_SQ_FT         INTEGER                       ,
    W_STREET_NUMBER           CHAR(10)                      ,
    W_STREET_NAME             VARCHAR(60)                   ,
    W_STREET_TYPE             CHAR(15)                      ,
    W_SUITE_NUMBER            CHAR(10)                      ,
    W_CITY                    VARCHAR(60)                   ,
    W_COUNTY                  VARCHAR(30)                   ,
    W_STATE                   CHAR(2)                       ,
    W_ZIP                     CHAR(10)                      ,
    W_COUNTRY                 VARCHAR(20)                   ,
    W_GMT_OFFSET              DECIMAL(5,2)
);

openGauss=# CREATE TABLE tpcds.warehouse_t2
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)                   ,
    W_WAREHOUSE_SQ_FT         INTEGER                       ,
    W_STREET_NUMBER           CHAR(10)                      ,
    W_STREET_NAME             VARCHAR(60),
    W_STREET_TYPE             CHAR(15)                      ,
    W_SUITE_NUMBER            CHAR(10)                      ,
    W_CITY                    VARCHAR(60)                   ,
    W_COUNTY                  VARCHAR(30)                   ,
    W_STATE                   CHAR(2)                       ,
    W_ZIP                     CHAR(10)                      ,
    W_COUNTRY                 VARCHAR(20)                   ,
    W_GMT_OFFSET              DECIMAL(5,2)
);
-- Create a table and set the default value of the W_STATE column to GA.
openGauss=# CREATE TABLE tpcds.warehouse_t3
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)                   ,
    W_WAREHOUSE_SQ_FT         INTEGER                       ,
    W_STREET_NUMBER           CHAR(10)                      ,
    W_STREET_NAME             VARCHAR(60)                   ,
    W_STREET_TYPE             CHAR(15)                      ,
    W_SUITE_NUMBER            CHAR(10)                      ,
    W_CITY                    VARCHAR(60)                   ,
    W_COUNTY                  VARCHAR(30)                   ,
    W_STATE                   CHAR(2)           DEFAULT 'GA',
    W_ZIP                     CHAR(10)                      ,
    W_COUNTRY                 VARCHAR(20)                   ,
    W_GMT_OFFSET              DECIMAL(5,2)
);

-- Create a table and check whether the W_WAREHOUSE_NAME column is unique at the end of its creation.
openGauss=# CREATE TABLE tpcds.warehouse_t4
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)   UNIQUE DEFERRABLE,
    W_WAREHOUSE_SQ_FT         INTEGER                       ,
    W_STREET_NUMBER           CHAR(10)                      ,
    W_STREET_NAME             VARCHAR(60)                   ,
    W_STREET_TYPE             CHAR(15)                      ,
    W_SUITE_NUMBER            CHAR(10)                      ,
    W_CITY                    VARCHAR(60)                   ,
    W_COUNTY                  VARCHAR(30)                   ,
    W_STATE                   CHAR(2)                       ,
    W_ZIP                     CHAR(10)                      ,
    W_COUNTRY                 VARCHAR(20)                   ,
    W_GMT_OFFSET              DECIMAL(5,2)
);
-- Create a table with its fill factor set to 70%.
openGauss=# CREATE TABLE tpcds.warehouse_t5
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)                   ,
    W_WAREHOUSE_SQ_FT         INTEGER                       ,
    W_STREET_NUMBER           CHAR(10)                      ,
    W_STREET_NAME             VARCHAR(60)                   ,
    W_STREET_TYPE             CHAR(15)                      ,
    W_SUITE_NUMBER            CHAR(10)                      ,
    W_CITY                    VARCHAR(60)                   ,
    W_COUNTY                  VARCHAR(30)                   ,
    W_STATE                   CHAR(2)                       ,
    W_ZIP                     CHAR(10)                      ,
    W_COUNTRY                 VARCHAR(20)                   ,
    W_GMT_OFFSET              DECIMAL(5,2),
    UNIQUE(W_WAREHOUSE_NAME) WITH(fillfactor=70)
);

-- Alternatively, user the following syntax:
openGauss=# CREATE TABLE tpcds.warehouse_t6
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)               UNIQUE,
    W_WAREHOUSE_SQ_FT         INTEGER                    ,
    W_STREET_NUMBER           CHAR(10)                   ,
    W_STREET_NAME             VARCHAR(60)                ,
    W_STREET_TYPE             CHAR(15)                   ,
    W_SUITE_NUMBER            CHAR(10)                   ,
    W_CITY                    VARCHAR(60)                ,
    W_COUNTY                  VARCHAR(30)                ,
    W_STATE                   CHAR(2)                    ,
    W_ZIP                     CHAR(10)                   ,
    W_COUNTRY                 VARCHAR(20)                ,
    W_GMT_OFFSET              DECIMAL(5,2)
) WITH(fillfactor=70);

-- Create a table and specify that its data is not written to WALs.
openGauss=# CREATE UNLOGGED TABLE tpcds.warehouse_t7
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)                ,
    W_WAREHOUSE_SQ_FT         INTEGER                    ,
    W_STREET_NUMBER           CHAR(10)                   ,
    W_STREET_NAME             VARCHAR(60)                ,
    W_STREET_TYPE             CHAR(15)                   ,
    W_SUITE_NUMBER            CHAR(10)                   ,
    W_CITY                    VARCHAR(60)                ,
    W_COUNTY                  VARCHAR(30)                ,
    W_STATE                   CHAR(2)                    ,
    W_ZIP                     CHAR(10)                   ,
    W_COUNTRY                 VARCHAR(20)                ,
    W_GMT_OFFSET              DECIMAL(5,2)
);

-- Create a temporary table.
openGauss=# CREATE TEMPORARY TABLE warehouse_t24
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)                ,
    W_WAREHOUSE_SQ_FT         INTEGER                    ,
    W_STREET_NUMBER           CHAR(10)                   ,
    W_STREET_NAME             VARCHAR(60)                ,
    W_STREET_TYPE             CHAR(15)                   ,
    W_SUITE_NUMBER            CHAR(10)                   ,
    W_CITY                    VARCHAR(60)                ,
    W_COUNTY                  VARCHAR(30)                ,
    W_STATE                   CHAR(2)                    ,
    W_ZIP                     CHAR(10)                   ,
    W_COUNTRY                 VARCHAR(20)                ,
    W_GMT_OFFSET              DECIMAL(5,2)
);

-- Create a local temporary table and specify that this table is dropped when the transaction is committed.
openGauss=# CREATE TEMPORARY TABLE warehouse_t25
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)                ,
    W_WAREHOUSE_SQ_FT         INTEGER                    ,
    W_STREET_NUMBER           CHAR(10)                   ,
    W_STREET_NAME             VARCHAR(60)                ,
    W_STREET_TYPE             CHAR(15)                   ,
    W_SUITE_NUMBER            CHAR(10)                   ,
    W_CITY                    VARCHAR(60)                ,
    W_COUNTY                  VARCHAR(30)                ,
    W_STATE                   CHAR(2)                    ,
    W_ZIP                     CHAR(10)                   ,
    W_COUNTRY                 VARCHAR(20)                ,
    W_GMT_OFFSET              DECIMAL(5,2)
```

```
) ON COMMIT DELETE ROWS;

-- Create a global temporary table and specify that the temporary table data is deleted when the session
ends. The current Ustore storage engine does not support global temporary tables.
openGauss=# CREATE GLOBAL TEMPORARY TABLE gtt1
(
    ID                        INTEGER               NOT NULL,
    NAME                      CHAR(16)              NOT NULL,
    ADDRESS                   VARCHAR(50)                   ,
    POSTCODE                  CHAR(6)
) ON COMMIT PRESERVE ROWS;

-- Create a table and specify that no error is reported for duplicate tables (if any).
openGauss=# CREATE TABLE IF NOT EXISTS tpcds.warehouse_t8
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)                   ,
    W_WAREHOUSE_SQ_FT         INTEGER                       ,
    W_STREET_NUMBER           CHAR(10)                      ,
    W_STREET_NAME             VARCHAR(60)                   ,
    W_STREET_TYPE             CHAR(15)                      ,
    W_SUITE_NUMBER            CHAR(10)                      ,
    W_CITY                    VARCHAR(60)                   ,
    W_COUNTY                  VARCHAR(30)                   ,
    W_STATE                   CHAR(2)                       ,
    W_ZIP                     CHAR(10)                      ,
    W_COUNTRY                 VARCHAR(20)                   ,
    W_GMT_OFFSET              DECIMAL(5,2)
);

-- Create a general tablespace.
openGauss=# CREATE TABLESPACE DS_TABLESPACE1 RELATIVE LOCATION 'tablespace/tablespace_1';
-- Specify a tablespace when creating a table.
openGauss=# CREATE TABLE tpcds.warehouse_t9
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)                   ,
    W_WAREHOUSE_SQ_FT         INTEGER                       ,
    W_STREET_NUMBER           CHAR(10)                      ,
    W_STREET_NAME             VARCHAR(60)                   ,
    W_STREET_TYPE             CHAR(15)                      ,
    W_SUITE_NUMBER            CHAR(10)                      ,
    W_CITY                    VARCHAR(60)                   ,
    W_COUNTY                  VARCHAR(30)                   ,
    W_STATE                   CHAR(2)                       ,
    W_ZIP                     CHAR(10)                      ,
    W_COUNTRY                 VARCHAR(20)                   ,
    W_GMT_OFFSET              DECIMAL(5,2)
) TABLESPACE DS_TABLESPACE1;

-- Separately specify the index tablespace for W_WAREHOUSE_NAME when creating the table.
openGauss=# CREATE TABLE tpcds.warehouse_t10
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)           UNIQUE USING INDEX TABLESPACE
DS_TABLESPACE1,
    W_WAREHOUSE_SQ_FT         INTEGER                       ,
    W_STREET_NUMBER           CHAR(10)                      ,
    W_STREET_NAME             VARCHAR(60)                   ,
    W_STREET_TYPE             CHAR(15)                      ,
    W_SUITE_NUMBER            CHAR(10)                      ,
    W_CITY                    VARCHAR(60)                   ,
    W_COUNTY                  VARCHAR(30)                   ,
    W_STATE                   CHAR(2)                       ,
    W_ZIP                     CHAR(10)                      ,
    W_COUNTRY                 VARCHAR(20)                   ,
    W_GMT_OFFSET              DECIMAL(5,2)
);
-- Create a table with a primary key constraint.
openGauss=# CREATE TABLE tpcds.warehouse_t11
(
    W_WAREHOUSE_SK            INTEGER            PRIMARY KEY,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)                   ,
    W_WAREHOUSE_SQ_FT         INTEGER                       ,
    W_STREET_NUMBER           CHAR(10)                      ,
    W_STREET_NAME             VARCHAR(60)                   ,
    W_STREET_TYPE             CHAR(15)                      ,
    W_SUITE_NUMBER            CHAR(10)                      ,
    W_CITY                    VARCHAR(60)                   ,
    W_COUNTY                  VARCHAR(30)                   ,
    W_STATE                   CHAR(2)                       ,
    W_ZIP                     CHAR(10)                      ,
    W_COUNTRY                 VARCHAR(20)                   ,
    W_GMT_OFFSET              DECIMAL(5,2)
);

-- An alternative for the preceding syntax is as follows:
openGauss=# CREATE TABLE tpcds.warehouse_t12
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)                   ,
    W_WAREHOUSE_SQ_FT         INTEGER                       ,
    W_STREET_NUMBER           CHAR(10)                      ,
    W_STREET_NAME             VARCHAR(60)                   ,
    W_STREET_TYPE             CHAR(15)                      ,
    W_SUITE_NUMBER            CHAR(10)                      ,
    W_CITY                    VARCHAR(60)                   ,
    W_COUNTY                  VARCHAR(30)                   ,
    W_STATE                   CHAR(2)                       ,
    W_ZIP                     CHAR(10)                      ,
    W_COUNTRY                 VARCHAR(20)                   ,
    W_GMT_OFFSET              DECIMAL(5,2),
    PRIMARY KEY(W_WAREHOUSE_SK)
);

-- Or use the following statement to specify the name of the constraint:
openGauss=# CREATE TABLE tpcds.warehouse_t13
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)                   ,
    W_WAREHOUSE_SQ_FT         INTEGER                       ,
    W_STREET_NUMBER           CHAR(10)                      ,
    W_STREET_NAME             VARCHAR(60)                   ,
    W_STREET_TYPE             CHAR(15)                      ,
    W_SUITE_NUMBER            CHAR(10)                      ,
    W_CITY                    VARCHAR(60)                   ,
    W_COUNTY                  VARCHAR(30)                   ,
    W_STATE                   CHAR(2)                       ,
    W_ZIP                     CHAR(10)                      ,
    W_COUNTRY                 VARCHAR(20)                   ,
    W_GMT_OFFSET              DECIMAL(5,2),
    CONSTRAINT W_CSTR_KEY1 PRIMARY KEY(W_WAREHOUSE_SK)
);

-- Create a table with a compound primary key constraint.
openGauss=# CREATE TABLE tpcds.warehouse_t14
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)                   ,
    W_WAREHOUSE_SQ_FT         INTEGER                       ,
    W_STREET_NUMBER           CHAR(10)                      ,
    W_STREET_NAME             VARCHAR(60)                   ,
    W_STREET_TYPE             CHAR(15)                      ,
    W_SUITE_NUMBER            CHAR(10)                      ,
    W_CITY                    VARCHAR(60)                   ,
    W_COUNTY                  VARCHAR(30)                   ,
    W_STATE                   CHAR(2)                       ,
    W_ZIP                     CHAR(10)                      ,
    W_COUNTRY                 VARCHAR(20)                   ,
    W_GMT_OFFSET              DECIMAL(5,2),
    CONSTRAINT W_CSTR_KEY2 PRIMARY KEY(W_WAREHOUSE_SK, W_WAREHOUSE_ID)
);
-- Create a column-store table.
openGauss=# CREATE TABLE tpcds.warehouse_t15
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)                   ,
    W_WAREHOUSE_SQ_FT         INTEGER                       ,
    W_STREET_NUMBER           CHAR(10)                      ,
    W_STREET_NAME             VARCHAR(60)                   ,
    W_STREET_TYPE             CHAR(15)                      ,
    W_SUITE_NUMBER            CHAR(10)                      ,
    W_CITY                    VARCHAR(60)                   ,
    W_COUNTY                  VARCHAR(30)                   ,
    W_STATE                   CHAR(2)                       ,
    W_ZIP                     CHAR(10)                      ,
    W_COUNTRY                 VARCHAR(20)                   ,
    W_GMT_OFFSET              DECIMAL(5,2)
) WITH (ORIENTATION = COLUMN);

-- Create a column-store table using partial clustered storage.
openGauss=# CREATE TABLE tpcds.warehouse_t16
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)                   ,
    W_WAREHOUSE_SQ_FT         INTEGER                       ,
    W_STREET_NUMBER           CHAR(10)                      ,
    W_STREET_NAME             VARCHAR(60)                   ,
    W_STREET_TYPE             CHAR(15)                      ,
    W_SUITE_NUMBER            CHAR(10)                      ,
    W_CITY                    VARCHAR(60)                   ,
    W_COUNTY                  VARCHAR(30)                   ,
    W_STATE                   CHAR(2)                       ,
    W_ZIP                     CHAR(10)                      ,
    W_COUNTRY                 VARCHAR(20)                   ,
    W_GMT_OFFSET              DECIMAL(5,2),
    PARTIAL CLUSTER KEY(W_WAREHOUSE_SK, W_WAREHOUSE_ID)
) WITH (ORIENTATION = COLUMN);

-- Define a column-store table with compression enabled.
openGauss=# CREATE TABLE tpcds.warehouse_t17
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)                   ,
    W_WAREHOUSE_SQ_FT         INTEGER                       ,
    W_STREET_NUMBER           CHAR(10)                      ,
    W_STREET_NAME             VARCHAR(60)                   ,
    W_STREET_TYPE             CHAR(15)                      ,
    W_SUITE_NUMBER            CHAR(10)                      ,
    W_CITY                    VARCHAR(60)                   ,
    W_COUNTY                  VARCHAR(30)                   ,
    W_STATE                   CHAR(2)                       ,
    W_ZIP                     CHAR(10)                      ,
    W_COUNTRY                 VARCHAR(20)                   ,
    W_GMT_OFFSET              DECIMAL(5,2)
) WITH (ORIENTATION = COLUMN, COMPRESSION=HIGH);
```

```
-- Define a column check constraint.
openGauss=# CREATE TABLE tpcds.warehouse_t19
(
    W_WAREHOUSE_SK            INTEGER               PRIMARY KEY CHECK (W_WAREHOUSE_SK > 0),
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)           CHECK (W_WAREHOUSE_NAME IS NOT NULL),
    W_WAREHOUSE_SQ_FT         INTEGER                       ,
    W_STREET_NUMBER           CHAR(10)                      ,
    W_STREET_NAME             VARCHAR(60)                   ,
    W_STREET_TYPE             CHAR(15)                      ,
    W_SUITE_NUMBER            CHAR(10)                      ,
    W_CITY                    VARCHAR(60)                   ,
    W_COUNTY                  VARCHAR(30)                   ,
    W_STATE                   CHAR(2)                       ,
    W_ZIP                     CHAR(10)                      ,
    W_COUNTRY                 VARCHAR(20)                   ,
    W_GMT_OFFSET              DECIMAL(5,2)
);

openGauss=# CREATE TABLE tpcds.warehouse_t20
(
    W_WAREHOUSE_SK            INTEGER               PRIMARY KEY,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)           CHECK (W_WAREHOUSE_NAME IS NOT NULL),
    W_WAREHOUSE_SQ_FT         INTEGER                       ,
    W_STREET_NUMBER           CHAR(10)                      ,
    W_STREET_NAME             VARCHAR(60)                   ,
    W_STREET_TYPE             CHAR(15)                      ,
    W_SUITE_NUMBER            CHAR(10)                      ,
    W_CITY                    VARCHAR(60)                   ,
    W_COUNTY                  VARCHAR(30)                   ,
    W_STATE                   CHAR(2)                       ,
    W_ZIP                     CHAR(10)                      ,
    W_COUNTRY                 VARCHAR(20)                   ,
    W_GMT_OFFSET              DECIMAL(5,2),
    CONSTRAINT W_CONSTR_KEY2 CHECK(W_WAREHOUSE_SK > 0 AND W_WAREHOUSE_NAME IS NOT 
NULL)
);

-- Create a table with a foreign key constraint.
openGauss=# CREATE TABLE tpcds.city_t23
(
    W_CITY            VARCHAR(60)                PRIMARY KEY,
    W_ADDRESS         TEXT
);
openGauss=# CREATE TABLE tpcds.warehouse_t23
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)                   ,
    W_WAREHOUSE_SQ_FT         INTEGER                       ,
    W_STREET_NUMBER           CHAR(10)                      ,
    W_STREET_NAME             VARCHAR(60)                   ,
    W_STREET_TYPE             CHAR(15)                      ,
    W_SUITE_NUMBER            CHAR(10)                      ,
    W_CITY                    VARCHAR(60)           REFERENCES tpcds.city_t23(W_CITY),
    W_COUNTY                  VARCHAR(30)                   ,
    W_STATE                   CHAR(2)                       ,
    W_ZIP                     CHAR(10)                      ,
    W_COUNTRY                 VARCHAR(20)                   ,
    W_GMT_OFFSET              DECIMAL(5,2)
);

-- An alternative for the preceding syntax is as follows:
openGauss=# CREATE TABLE tpcds.warehouse_t23
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)                   ,
    W_WAREHOUSE_SQ_FT         INTEGER                       ,
    W_STREET_NUMBER           CHAR(10)                      ,
    W_STREET_NAME             VARCHAR(60)                   ,
    W_STREET_TYPE             CHAR(15)                      ,
    W_SUITE_NUMBER            CHAR(10)                      ,
    W_CITY                    VARCHAR(60)                   ,
    W_COUNTY                  VARCHAR(30)                   ,
    W_STATE                   CHAR(2)                       ,
    W_ZIP                     CHAR(10)                      ,
    W_COUNTRY                 VARCHAR(20)                   ,
    W_GMT_OFFSET              DECIMAL(5,2)                  ,
    FOREIGN KEY(W_CITY) REFERENCES tpcds.city_t23(W_CITY)
);

-- Or use the following statement to specify the name of the constraint:
openGauss=# CREATE TABLE tpcds.warehouse_t23
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)                   ,
    W_WAREHOUSE_SQ_FT         INTEGER                       ,
    W_STREET_NUMBER           CHAR(10)                      ,
    W_STREET_NAME             VARCHAR(60)                   ,
    W_STREET_TYPE             CHAR(15)                      ,
    W_SUITE_NUMBER            CHAR(10)                      ,
    W_CITY                    VARCHAR(60)                   ,
    W_COUNTY                  VARCHAR(30)                   ,
    W_STATE                   CHAR(2)                       ,
    W_ZIP                     CHAR(10)                      ,
    W_COUNTRY                 VARCHAR(20)                   ,
    W_GMT_OFFSET              DECIMAL(5,2)                  ,
    CONSTRAINT W_FORE_KEY1 FOREIGN KEY(W_CITY) REFERENCES tpcds.city_t23(W_CITY)
);

-- Add a varchar column to the tpcds.warehouse_t19 table.
openGauss=# ALTER TABLE tpcds.warehouse_t19 ADD W_GOODS_CATEGORY varchar(30);

-- Add a check constraint to the tpcds.warehouse_t19 table.
openGauss=# ALTER TABLE tpcds.warehouse_t19 ADD CONSTRAINT W_CONSTR_KEY4 CHECK (W_STATE IS
NOT NULL);

-- Use one statement to alter the types of two existing columns.
openGauss=# ALTER TABLE tpcds.warehouse_t19
    ALTER COLUMN W_GOODS_CATEGORY TYPE varchar(80),
    ALTER COLUMN W_STREET_NAME TYPE varchar(100);

-- This statement is equivalent to the preceding statement.
openGauss=# ALTER TABLE tpcds.warehouse_t19 MODIFY (W_GOODS_CATEGORY varchar(30),
W_STREET_NAME varchar(60));

-- Add a not-null constraint to an existing column.
openGauss=# ALTER TABLE tpcds.warehouse_t19 ALTER COLUMN W_GOODS_CATEGORY SET NOT NULL;

-- Remove not-null constraints from an existing column.
openGauss=# ALTER TABLE tpcds.warehouse_t19 ALTER COLUMN W_GOODS_CATEGORY DROP NOT NULL;

-- If no partial cluster is specified in a column-store table, add a partial cluster to the table.
openGauss=# ALTER TABLE tpcds.warehouse_t17 ADD PARTIAL CLUSTER KEY(W_WAREHOUSE_SK);

-- View the constraint name and delete the partial cluster column of a column-store table.
openGauss=# \d+ tpcds.warehouse_t17
                              Table "tpcds.warehouse_t17"
       Column       |         Type          | Modifiers | Storage  | Stats target | Description
-------------------+-----------------------+-----------+----------+--------------+-------------
 w_warehouse_sk     | integer               | not null  | plain    |              |
 w_warehouse_id     | character(16)         | not null  | extended |              |
 w_warehouse_name   | character varying(20) |           | extended |              |
 w_warehouse_sq_ft  | integer               |           | plain    |              |
 w_street_number   | character(10)         |           | extended |              |
 w_street_name     | character varying(60) |           | extended |              |
 w_street_type     | character(15)         |           | extended |              |
 w_suite_number    | character(10)         |           | extended |              |
 w_city            | character varying(60) |           | extended |              |
 w_county          | character varying(30) |           | extended |              |
 w_state           | character(2)          |           | extended |              |
 w_zip             | character(10)         |           | extended |              |
 w_country         | character varying(20) |           | extended |              |
 w_gmt_offset      | numeric(5,2)          |           | main     |              |
Partial Cluster :
    "warehouse_t17_cluster" PARTIAL CLUSTER KEY (w_warehouse_sk)
Has OIDs: no
Location Nodes: ALL DATANODES
Options: compression=no, version=0.12
openGauss=# ALTER TABLE tpcds.warehouse_t17 DROP CONSTRAINT warehouse_t17_cluster;

-- Move a table to another tablespace.
openGauss=# ALTER TABLE tpcds.warehouse_t19 SET TABLESPACE PG_DEFAULT;
-- Create the joe schema.
openGauss=# CREATE SCHEMA joe;

-- Move a table to another schema.
openGauss=# ALTER TABLE tpcds.warehouse_t19 SET SCHEMA joe;

-- Rename an existing table.
openGauss=# ALTER TABLE joe.warehouse_t19 RENAME TO warehouse_t23;

-- Delete a column from the warehouse_t23 table.
openGauss=# ALTER TABLE joe.warehouse_t23 DROP COLUMN W_STREET_NAME;

-- Delete the tablespace, schema joe, and schema tables warehouse.
openGauss=# DROP TABLE tpcds.warehouse_t1;
openGauss=# DROP TABLE tpcds.warehouse_t2;
openGauss=# DROP TABLE tpcds.warehouse_t3;
openGauss=# DROP TABLE tpcds.warehouse_t4;
openGauss=# DROP TABLE tpcds.warehouse_t5;
openGauss=# DROP TABLE tpcds.warehouse_t6;
openGauss=# DROP TABLE tpcds.warehouse_t7;
openGauss=# DROP TABLE tpcds.warehouse_t8;
openGauss=# DROP TABLE tpcds.warehouse_t9;
openGauss=# DROP TABLE tpcds.warehouse_t10;
openGauss=# DROP TABLE tpcds.warehouse_t11;
openGauss=# DROP TABLE tpcds.warehouse_t12;
openGauss=# DROP TABLE tpcds.warehouse_t13;
openGauss=# DROP TABLE tpcds.warehouse_t14;
openGauss=# DROP TABLE tpcds.warehouse_t15;
openGauss=# DROP TABLE tpcds.warehouse_t16;
openGauss=# DROP TABLE tpcds.warehouse_t17;
openGauss=# DROP TABLE tpcds.warehouse_t18;
openGauss=# DROP TABLE tpcds.warehouse_t20;
openGauss=# DROP TABLE tpcds.warehouse_t21;
openGauss=# DROP TABLE tpcds.warehouse_t22;
openGauss=# DROP TABLE joe.warehouse_t23;
openGauss=# DROP TABLE tpcds.warehouse_t24;
openGauss=# DROP TABLE tpcds.warehouse_t25;
openGauss=# DROP TABLESPACE DS_TABLESPACE1;
openGauss=# DROP SCHEMA IF EXISTS joe CASCADE;
```

Helpful Links

ALTER TABLE, **DROP TABLE**, and **CREATE TABLESPACE**.

Suggestions

- UNLOGGED

  - The unlogged table and its indexes do not use the WAL log mechanism during data writing. Their write speed is much higher than that of ordinary tables. Therefore, they can be used for storing intermediate result sets of complex queries to improve query performance.
  - The unlogged table has no primary/standby mechanism. In case of system faults or abnormal breakpoints, data loss may occur. Therefore, the unlogged table cannot be used to store basic data.
- TEMPORARY | TEMP
  - A temporary table is automatically dropped at the end of a session.
- LIKE
  - The new table automatically inherits all column names, data types, and not-null constraints from this table. The new table is irrelevant to the original table after the creation.
- LIKE INCLUDING DEFAULTS
  - The default expressions are copied from the original table to the new table only if **INCLUDING DEFAULTS** is specified. The default behavior is to exclude default expressions, resulting in the copied columns in the new table having default values **NULL**.
- LIKE INCLUDING CONSTRAINTS
  - The **CHECK** constraints are copied from the original table to the new table only when **INCLUDING CONSTRAINTS** is specified. Other types of constraints are never copied to the new table. Not-null constraints are always copied to the new table. These rules also apply to column constraints and table constraints.
- LIKE INCLUDING INDEXES
  - Any indexes on the original table will not be created on the new table, unless the **INCLUDING INDEXES** clause is specified.
- LIKE INCLUDING STORAGE
  - **STORAGE** settings for the copied column definitions are copied only if **INCLUDING STORAGE** is specified. The default behavior is to exclude **STORAGE** settings.
- LIKE INCLUDING COMMENTS
  - If **INCLUDING COMMENTS** is specified, comments for the copied columns, constraints, and indexes are copied. The default behavior is to exclude comments.
- LIKE INCLUDING PARTITION
  - If **INCLUDING PARTITION** is specified, the partition definitions of the source table are copied to the new table, and the new table no longer uses the **PARTITION BY** clause. The default behavior is to exclude partition definition of the original table.

NOTICE

List and hash partitioned tables do not support **LIKE INCLUDING PARTITION**.

- LIKE INCLUDING RELOPTIONS

  – If **INCLUDING RELOPTIONS** is specified, the new table will copy the storage parameter (that is, **WITH** clause) of the source table. The default behavior is to exclude partition definition of the storage parameter of the original table.
- LIKE INCLUDING ALL
  – **INCLUDING ALL** contains the meaning of **INCLUDING DEFAULTS**, **INCLUDING CONSTRAINTS**, **INCLUDING INDEXES**, **INCLUDING STORAGE**, **INCLUDING COMMENTS**, **INCLUDING PARTITION**, and **INCLUDING RELOPTIONS**.
- ORIENTATION ROW
  – Creates a row-store table. Row-store applies to the OLTP service, which has many interactive transactions. An interaction involves many columns in the table. Using row-store can improve the efficiency.
- ORIENTATION COLUMN
  – Creates a column-store table. Column-store applies to the DWS, which has a large amount of aggregation computing, and involves a few column operations.

11.14.87 CREATE TABLE AS

Function

CREATE TABLE AS creates a table from the results of a query.

It creates a table and fills it with data obtained using **SELECT**. The table columns have the names and data types associated with the output columns of **SELECT** (except that you can override the **SELECT** output column names by giving an explicit list of new column names).

CREATE TABLE AS queries a source table once and writes the data in a new table. The result in the query view changes with the source table. In contrast, the view re-computes and defines its **SELECT** statement at each query.

Precautions

- This statement cannot be used to create a partitioned table.
- If an error occurs during table creation, after it is fixed, the system may fail to delete the disk files that are created before the last automatic clearance and whose size is not 0. This problem seldom occurs and does not affect system running of the database.

Syntax

```
CREATE [ [ GLOBAL | LOCAL ] [ TEMPORARY | TEMP ] | UNLOGGED ] TABLE table_name
    [ (column_name [, ...] ) ]
    [ WITH ( {storage_parameter = value} [, ... ] ) ]
    [ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
    [ COMPRESS | NOCOMPRESS ]
    [ TABLESPACE tablespace_name ]
    AS query
    [ WITH [ NO ] DATA ];
```

Parameter Description

- **UNLOGGED**

  Specifies that the table is created as an unlogged table. Data written to unlogged tables is not written to the WALs, which makes them considerably faster than ordinary tables. However, they are not crash-safe: an unlogged table is automatically truncated after a crash or unclean shutdown. Contents of an unlogged table are also not replicated to standby servers. Any indexes created on an unlogged table are automatically unlogged as well.

  – Usage scenario: Unlogged tables do not ensure data security. Users can back up data before using unlogged tables; for example, users should back up the data before a system upgrade.

  – Troubleshooting: If data is missing in the indexes of unlogged tables due to some unexpected operations such as an unclean shutdown, users should re-create the indexes with errors.

- **GLOBAL | LOCAL**

  When creating a temporary table, you can specify the **GLOBAL** or **LOCAL** keyword before **TEMP** or **TEMPORARY**. If the keyword **GLOBAL** is specified, GaussDB creates a global temporary table. Otherwise, GaussDB creates a local temporary table.

- **TEMPORARY | TEMP**

  If **TEMP** or **TEMPORARY** is specified, the created table is a temporary table. Temporary tables are classified into global temporary tables and local temporary tables. If the keyword **GLOBAL** is specified when a temporary table is created, the table is a global temporary table. Otherwise, the table is a local temporary table.

  The metadata of the global temporary table is visible to all sessions. After the sessions end, the metadata still exists. The user data, indexes, and statistics of a session are isolated from those of another session. Each session can only view and modify the data submitted by itself. Global temporary tables have two schemas: **ON COMMIT PRESERVE ROWS** and **ON COMMIT PRESERVE ROWS**. In session-based **ON COMMIT PRESERVE ROWS** schema, user data is automatically cleared when a session ends. In transaction-based **ON COMMIT DELETE ROWS** schema, user data is automatically cleared when the commit or rollback operation is performed. If the **ON COMMIT** option is not specified during table creation, the session level is used by default. Different from local temporary tables, you can specify a schema that does not start with **pg_temp_** when creating a global temporary table.

  A local temporary table is automatically dropped at the end of the current session. Therefore, you can create and use temporary tables in the current session as long as the connected database node in the session is normal. Temporary tables are created only in the current session. If a DDL statement involves operations on temporary tables, a DDL error will be generated. Therefore, you are not advised to perform operations on temporary tables in DDL statements. **TEMP** is equivalent to **TEMPORARY**.

NOTICE

- Local temporary tables are visible to the current session through the schema starting with **pg_temp** start. Users should not delete schema started with **pg_temp** or **pg_toast_temp**.
- If **TEMPORARY** or **TEMP** is not specified when you create a table but its schema is set to that starting with **pg_temp_** in the current session, the table will be created as a temporary table.
- If global temporary tables and indexes are being used by other sessions, do not perform **ALTER** or **DROP**.
- The DDL of a global temporary table affects only the user data and indexes of the current session. For example, **TRUNCATE**, **REINDEX**, and **ANALYZE** are valid only for the current session.

- **table_name**

  Specifies the name of the table to be created.

  Value range: a string. It must comply with the naming convention.

- **column_name**

  Specifies the name of a column to be created in the new table.

  Value range: a string. It must comply with the naming convention.

- **WITH (storage_parameter [= value] [, ...])**

  Specifies an optional storage parameter for a table or an index. See details of parameters below.

  - FILLFACTOR

    The fill factor of a table is a percentage from 10 to 100. **100** (complete filling) is the default value. When a smaller fill factor is specified, **INSERT** operations pack table pages only to the indicated percentage. The remaining space on each page is reserved for updating rows on that page. This gives **UPDATE** a chance to place the updated copy of a row on the same page, which is more efficient than placing it on a different page. For a table whose entries are never updated, setting the fill factor to **100** (complete filling) is the best choice, but in heavily updated tables a smaller fill factor would be appropriate. The parameter is only valid for row-store tables.

    Value range: 10–100

  - ORIENTATION

    Value range:

    COLUMN: The data will be stored in columns.

    ROW (default value): The data will be stored in rows.

  - COMPRESSION

    Specifies the compression level of table data. It determines the compression ratio and time. Generally, the higher the level of compression, the higher the ratio, the longer the time; and the lower the level of compression, the lower the ratio, the shorter the time. The actual compression ratio depends on the distribution mode of table data loaded.

    Value range:

The valid values for column-store tables are **YES**, **NO**, **LOW**, **MIDDLE**, and **HIGH**, and the default value is **LOW**.

Row-store tables do not support compression.

- MAX_BATCHROW

  Specifies the maximum number of rows in a storage unit during data loading. The parameter is only valid for column-store tables.

  Value range: 10000 to 60000

- **ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP }**

  ON COMMIT determines what to do when you commit a temporary table creation operation. The three options are as follows. Currently, only **PRESERVE ROWS** and **DELETE ROWS** can be used.

  - **PRESERVE ROWS** (default): No special action is taken at the ends of transactions. The temporary table and its table data are unchanged.
  - **DELETE ROWS**: All rows in the temporary table will be deleted at the end of each transaction block.
  - **DROP**: The temporary table will be dropped at the end of the current transaction block. Only local temporary tables can be dropped. Global temporary tables cannot be dropped.

- **COMPRESS / NOCOMPRESS**

  Specifies keyword **COMPRESS** during the creation of a table, so that the compression feature is triggered in case of bulk **INSERT** operations. If this feature is enabled, a scan is performed for all tuple data within the page to generate a dictionary and then the tuple data is compressed and stored. If **NOCOMPRESS** is specified, the table is not compressed. Row-store tables do not support compression.

  Default value: **NOCOMPRESS**, that is, tuple data is not compressed before storage.

- **TABLESPACE tablespace_name**

  Specifies that the new table will be created in the **tablespace_name** tablespace. If not specified, the default tablespace is used.

- **AS query**

  Specifies a **SELECT** or **VALUES** command, or an **EXECUTE** command that runs a prepared **SELECT** or **VALUES** query.

- **[WITH [NO] DATA]**

  Specifies whether the data produced by the query should be copied to the new table. By default, the data will be copied. If the value **NO** is used, only the table structure will be copied.

Examples

```
-- Create the tpcds.store_returns table.
openGauss=# CREATE TABLE tpcds.store_returns
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    sr_item_sk                VARCHAR(20)                   ,
    W_WAREHOUSE_SQ_FT         INTEGER
);
-- Create the tpcds.store_returns_t1 table and insert numbers that are greater than 16 in the sr_item_sk column of the tpcds.store_returns table.
```

```
openGauss=# CREATE TABLE tpcds.store_returns_t1 AS SELECT * FROM tpcds.store_returns WHERE
sr_item_sk > '4795';

-- Copy tpcds.store_returns to create the tpcds.store_returns_t2 table.
openGauss=# CREATE TABLE tpcds.store_returns_t2 AS table tpcds.store_returns;

-- Delete the table.
openGauss=# DROP TABLE tpcds.store_returns_t1 ;
openGauss=# DROP TABLE tpcds.store_returns_t2 ;
openGauss=# DROP TABLE tpcds.store_returns;
```

Helpful Links

CREATE TABLE and **SELECT**

11.14.88 CREATE TABLE PARTITION

Function

CREATE TABLE PARTITION creates a partitioned table. Partitioning refers to splitting what is logically one large table into smaller physical pieces based on specific schemes. The table based on the logic is called a partitioned table, and each physical piece is called a partition. Data is stored on these physical partitions, instead of the logical partitioned table.

The common forms of partitioning include range partitioning, interval partitioning, hash partitioning, list partitioning, and value partitioning. Currently, row-store tables support range partitioning, interval partitioning, hash partitioning, and list partitioning. Column-store tables support only range partitioning.

In range partitioning, a table is partitioned based on ranges defined by one or more columns, with no overlap between the ranges of values assigned to different partitions. Each range has a dedicated partition for data storage.

The partitioning policy for range partitioning refers to how data is inserted into partitions. Currently, range partitioning only allows the use of the range partitioning policy.

In range partitioning, a table is partitioned based on partition key values. If a record can be mapped to a partition, it is inserted into the partition; if it cannot, an error message is returned. Range partitioning is the most commonly used partitioning policy.

Interval partitioning is a special type of range partitioning. Compared with range partitioning, interval value definition is added. When no matching partition can be found for an inserted record, a partition can be automatically created based on the interval value.

Interval partitioning supports only table-based partitioning of a list where the data type can be TIMESTAMP[(p)] [WITHOUT TIME ZONE], TIMESTAMP[(p)] [WITH TIME ZONE] and DATE.

Interval partitioning policy: A record is mapped to a created partition based on the partition key value. If the record can be mapped to a created partition, the record is inserted into the corresponding partition. Otherwise, a partition is automatically created based on the partition key value and table definition information, and then the record is inserted into the new partition. The data range of the new partition is equal to the interval value.

In hash partitioning, a modulus and a remainder are specified for each partition based on a column in the table, and records to be inserted into the table are allocated to the corresponding partition, the rows in each partition must meet the following condition: The value of the partition key divided by the specified modulus generates the remainder specified for the partition key.

In hash partitioning, table is partitioned based on partition key values. If a record can be mapped to a partition, it is inserted into the partition; if it cannot, an error message is returned.

List partitioning is to allocate the records to be inserted into a table to the corresponding partition based on the key values in each partition. The key values do not overlap in different partitions. Create a partition for each group of key values to store corresponding data.

In list partitioning, table is partitioned based on partition key values. If a record can be mapped to a partition, it is inserted into the partition; if it cannot, an error message is returned.

Partitioning can provide several benefits:

- Query performance can be improved drastically in certain situations, particularly when most of the heavily accessed rows of the table are in a single partition or a small number of partitions. Partitioning narrows the range of data search and improves data access efficiency.
- In the case of an INSERT or UPDATE operation on most portions of a single partition, performance can be improved by taking advantage of continuous scan of that partition instead of partitions scattered across the whole table.
- Frequent loading or deletion operations on records in a separate partition can be accomplished by reading or deleting that partition. This not only improves performance but also avoids the VACUUM overload caused by bulk DELETE operations (hash partitions cannot be deleted).

Precautions

- If the constraint key of the unique constraint and primary key constraint contains all partition keys, a local index is created for the constraints. Otherwise, a global index is created.
- Currently, hash partitioning and list partitioning support only single-column partitioning, and do not support multi-column partitioning.
- When you have the INSERT permission on an interval partitioned table, partitions can be automatically created when you run INSERT to write data to the table.
- In the PARTITION FOR (values) syntax for partitioned tables, values can only be constants.
- In the PARTITION FOR (values) syntax for partitioned tables, if data type conversion is required for values, you are advised to use forcible type conversion to prevent the implicit type conversion result from being inconsistent with the expected result.
- The maximum number of partitions is 1048575. Generally, it is impossible to create so many partitions, because too many partitions may cause insufficient memory. Create partitions based on the value of **local_syscache_threshold**. The memory used by the partitioned tables is about (number of partitions x

3/1024) MB. Theoretically, the memory occupied by the partitions cannot be greater than the value of **local_syscache_threshold**. In addition, some space must be reserved for other functions.
- Currently, the statement specifying a partition cannot perform global index scan.

Syntax

```
CREATE TABLE [ IF NOT EXISTS ] partition_table_name
( [
    { column_name data_type [ COLLATE collation ] [ column_constraint [ ... ] ]
    | table_constraint
    | LIKE source_table [ like_option [...] ] }[, ... ]
] )
    [ WITH ( {storage_parameter = value} [, ... ] ) ]
    [ COMPRESS | NOCOMPRESS ]
    [ TABLESPACE tablespace_name ]
     PARTITION BY {
        {RANGE (partition_key) [ INTERVAL ('interval_expr') [ STORE IN (tablespace_name [, ... ] ) ] ]
( partition_less_than_item [, ... ] )} |
        {RANGE (partition_key) [ INTERVAL ('interval_expr') [ STORE IN (tablespace_name [, ... ] ) ] ]
( partition_start_end_item [, ... ] )} |
        {LIST (partition_key) ( PARTITION partition_name VALUES (list_values) [TABLESPACE
tablespace_name][, ... ])} |
        {HASH (partition_key) ( PARTITION partition_name [TABLESPACE tablespace_name][, ... ])}
    } [ { ENABLE | DISABLE } ROW MOVEMENT ];
```

- **column_constraint** is as follows:

```
[ CONSTRAINT constraint_name ]
{ NOT NULL |
  NULL |
  CHECK ( expression ) |
  DEFAULT default_e xpr |
  GENERATED ALWAYS AS ( generation_expr ) STORED |
  UNIQUE index_parameters |
  PRIMARY KEY index_parameters |
  REFERENCES reftable [ ( refcolumn ) ] [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ]
        [ ON DELETE action ] [ ON UPDATE action ] }
[ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

- **table_constraint** is as follows:

```
[ CONSTRAINT constraint_name ]
{ CHECK ( expression ) |
  UNIQUE ( column_name [, ... ] ) index_parameters |
  PRIMARY KEY ( column_name [, ... ] ) index_parameters |
  FOREIGN KEY ( column_name [, ... ] ) REFERENCES reftable [ ( refcolumn [, ... ] ) ]
      [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ] [ ON DELETE action ] [ ON UPDATE
action ] }
[ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

- **like_option** is as follows:

```
{ INCLUDING | EXCLUDING } { DEFAULTS | GENERATED | CONSTRAINTS | INDEXES | STORAGE |
COMMENTS | RELOPTIONS| ALL }
```

- **index_parameters** is as follows:

```
[ WITH ( {storage_parameter = value} [, ... ] ) ]
[ USING INDEX TABLESPACE tablespace_name ]
```

- partition_less_than_item:

```
PARTITION partition_name VALUES LESS THAN ( { partition_value | MAXVALUE } ) [TABLESPACE
tablespace_name]
```

- partition_start_end_item:

```
PARTITION partition_name {
        {START(partition_value) END (partition_value) EVERY (interval_value)} |
        {START(partition_value) END ({partition_value | MAXVALUE})} |
        {START(partition_value)} |
        {END({partition_value | MAXVALUE})}
} [TABLESPACE tablespace_name]
```

Parameter Description

- **IF NOT EXISTS**

  Sends a notice, but does not throw an error, if a table with the same name exists.

- **partition_table_name**

  Specifies the name of a partitioned table.

  Value range: a string. It must comply with the identifier naming convention.

- **column_name**

  Specifies the name of a column to be created in the new table.

  Value range: a string. It must comply with the identifier naming convention.

- **data_type**

  Specifies the data type of the column.

- **COLLATE collation**

  Assigns a collation to the column (which must be of a collatable data type). If no collation is specified, the default collation is used. You can run the **select * from pg_collation;** command to query collation rules from the **pg_collation** system catalog. The default collation rule is the row starting with **default** in the query result.

- **CONSTRAINT constraint_name**

  Specifies the name of a column or table constraint. The optional constraint clauses specify constraints that new or updated rows must satisfy for an INSERT or UPDATE operation to succeed.

  There are two ways to define constraints:

  - A column constraint is defined as part of a column definition, and it is bound to a particular column.
  - A table constraint is not bound to a particular column but can apply to more than one column.

- **LIKE source_table [like_option ...]**

  Specifies a table from which the new table automatically copies all column names, their data types, and their not-null constraints.

  Unlike **INHERITS**, the new table and original table are decoupled after creation is complete. Changes to the original table will not be applied to the new table, and it is not possible to include data of the new table in scans of the original table.

  - Default expressions for the copied column definitions will be copied only if **INCLUDING DEFAULTS** is specified. The default behavior is to exclude default expressions, resulting in the copied columns in the new table having default values **NULL**.
  - If **INCLUDING GENERATED** is specified, the generated expression of the source table column is copied to the new table. By default, the generated expression is not copied.
  - Not-null constraints are always copied to the new table. **CHECK** constraints will only be copied if **INCLUDING CONSTRAINTS** is specified; other types of constraints will never be copied. These rules also apply to column constraints and table constraints.

  - Unlike those of **INHERITS**, columns and constraints copied by **LIKE** are not merged with similarly named columns and constraints. If the same name is specified explicitly or in another **LIKE** clause, an error is reported.
  - Any indexes on the original table will not be created on the new table, unless the **INCLUDING INDEXES** clause is specified.
  - **STORAGE** settings for the copied column definitions are copied only if **INCLUDING STORAGE** is specified. The default behavior is to exclude **STORAGE** settings.
  - If **INCLUDING COMMENTS** is specified, comments for the copied columns, constraints, and indexes are copied. The default behavior is to exclude comments.
  - If **INCLUDING RELOPTIONS** is specified, the new table will copy the storage parameter (that is, **WITH** clause) of the source table. The default behavior is to exclude partition definition of the storage parameter of the original table.
  - **INCLUDING ALL** contains the meaning of **INCLUDING DEFAULTS**, **INCLUDING CONSTRAINTS**, **INCLUDING INDEXES**, **INCLUDING STORAGE**, **INCLUDING COMMENTS**, **INCLUDING PARTITION**, and **INCLUDING RELOPTIONS**.

- **WITH (storage_parameter [= value] [, ...])**

  Specifies an optional storage parameter for a table or an index. Optional parameters are as follows:

  - FILLFACTOR

    The fill factor of a table is a percentage from 10 to 100. **100** (complete filling) is the default value. When a smaller fill factor is specified, **INSERT** operations fill table pages only to the indicated percentage. The remaining space on each page is reserved for updating rows on that page. This gives **UPDATE** a chance to place the updated copy of a row on the same page, which is more efficient than placing it on a different page. For a table whose entries are never updated, setting the fill factor to **100** (complete filling) is the best choice, but in heavily updated tables a smaller fill factor would be appropriate. The parameter has no meaning for column-store tables.

    Value range: 10–100

  - ORIENTATION

    Determines the storage mode of the data in the table.

    Value range:

    - **COLUMN**: The data will be stored in columns.
    - **ROW** (default value): The data will be stored in rows.

    NOTICE

    orientation cannot be modified.

  - STORAGE_TYPE

    Specifies the storage engine type. This parameter cannot be modified once it is set.

Value range:

  - **USTORE** indicates that tables support the inplace-update storage engine. Note that the **track_counts** and **track_activities** parameters must be enabled when the Ustore table is used. Otherwise, space expansion may occur.
  - **ASTORE** indicates that tables support the append-only storage engine.

Default value:

If no table is specified, data is stored in append-only mode by default.

- COMPRESSION
  - Valid values for column-store tables are **LOW**, **MIDDLE**, **HIGH**, **YES**, and **NO**, and the compression level increases accordingly. The default is **LOW**.
  - Row-store tables do not support compression.
- MAX_BATCHROW

  Specifies the maximum number of records in a storage unit during data loading. The parameter is only valid for column-store tables.

  Value range: 10000 to 60000. The default value is **60000**.
- PARTIAL_CLUSTER_ROWS

  Specifies the number of records to be partially clustered for storage during data loading. The parameter is only valid for column-store tables.

  Value range: greater than or equal to **MAX_BATCHROW**. You are advised to set this parameter to an integer multiple of **MAX_BATCHROW**.
- DELTAROW_THRESHOLD

  A reserved parameter. The parameter is only valid for column-store tables.

  Value range: 0 to 9999
- segment

  The data is stored in segment-page mode. This parameter supports only row-store tables. Column-store tables, temporary tables, and unlogged tables are not supported. The Ustore storage engine is not supported.

  Value range: **on** and **off**

  Default value: **off**

- **COMPRESS / NOCOMPRESS**

  Specifies keyword COMPRESS during the creation of a table, so that the compression feature is triggered in case of BULK INSERT operations. If this feature is enabled, a scan is performed for all tuple data within the page to generate a dictionary and then the tuple data is compressed and stored. If **NOCOMPRESS** is specified, the table is not compressed. Row-store tables do not support compression.

  Default value: **NOCOMPRESS**, that is, tuple data is not compressed before storage.
- **TABLESPACE tablespace_name**

Specifies that the new table will be created in the **tablespace_name** tablespace. If not specified, the default tablespace is used.

- **PARTITION BY RANGE(partition_key)**

  Creates a range partition. **partition_key** is the name of the partition key.

  (1) Assume that the **VALUES LESS THAN** syntax is used.

  NOTICE

  In this case, a maximum of four partition keys are supported.

  Data types supported by the partition keys are as follows: SMALLINT, INTEGER, BIGINT, DECIMAL, NUMERIC, REAL, DOUBLE PRECISION, CHARACTER VARYING(*n*), VARCHAR(*n*), CHARACTER(*n*), CHAR(*n*), CHARACTER, CHAR, TEXT, NVARCHAR, NVARCHAR2, NAME, TIMESTAMP[(p)] [WITHOUT TIME ZONE], TIMESTAMP[(p)] [WITH TIME ZONE], and DATE.

  (2) Assume that the **START END** syntax is used.

  NOTICE

  In this case, only one partition key is supported.

  Data types supported by the partition key are as follows: **SMALLINT**, **INTEGER**, **BIGINT**, **DECIMAL**, **NUMERIC**, **REAL**, **DOUBLE PRECISION**, **TIMESTAMP[(p)] [WITHOUT TIME ZONE]**, **TIMESTAMP[(p)] [WITH TIME ZONE]**, and **DATE**.

  (3) Assume that the **INTERVAL** syntax is used.

  NOTICE

  In this case, only one partition key is supported.

  In this case, the data types supported by the partition key are TIMESTAMP[(p)] [WITHOUT TIME ZONE], TIMESTAMP[(p)] [WITH TIME ZONE] and DATE.

- **PARTITION partition_name VALUES LESS THAN ({ partition_value | MAXVALUE })**

  Specifies the information of partitions. **partition_name** is the name of a range partition. **partition_value** is the upper limit of a range partition, and the value depends on the type of **partition_key**. *MAXVALUE* usually specifies the upper limit of the last range partition.

NOTICE

- Each partition requires an upper limit.
- The data type of the upper limit must be the same as that of the partition key.
- In a partition list, partitions are arranged in ascending order of upper limits. A partition with a smaller upper limit value is placed before another partition with a larger one.

- **PARTITION partition_name {START (partition_value) END (partition_value) EVERY (interval_value)} | {START (partition_value) END (partition_value|MAXVALUE)} | {START(partition_value)} | {END (partition_value | MAXVALUE)}**

  Specifies the information of partitions.

  – **partition_name**: name or name prefix of a range partition. It is the name prefix only in the following cases (assuming that **partition_name** is **p1**):

    ▪ If **START**+**END**+**EVERY** is used, the names of partitions will be defined as **p1_1**, **p1_2**, and the like. For example, if **PARTITION p1 START(1) END(4) EVERY(1)** is defined, the generated partitions are [1, 2), [2, 3), and [3, 4), and their names are **p1_1**, **p1_2**, and **p1_3**. In this case, **p1** is a name prefix.

    ▪ If the defined statement is in the first place and has **START** specified, the range (*MINVALUE*, **START**) will be automatically used as the first actual partition, and its name will be **p1_0**. The other partitions are then named **p1_1**, **p1_2**, and the like. For example, if **PARTITION p1 START(1), PARTITION p2 START(2)** is defined, generated partitions are (*MINVALUE*, 1), [1, 2), and [2, *MAXVALUE*), and their names will be **p1_0**, **p1_1**, and **p2**. In this case, **p1** is a name prefix and **p2** is a partition name. **MINVALUE** means the minimum value.

  – **partition_value**: start value or end value of a range partition. The value depends on **partition_key** and cannot be *MAXVALUE*.

  – **interval_value**: width of each partition for dividing the [**START**, **END**) range. It cannot be *MAXVALUE*. If the value of (**END** – **START**) divided by **EVERY** has a remainder, the width of only the last partition is less than the value of **EVERY**.

  – *MAXVALUE* usually specifies the upper limit of the last range partition.

NOTICE

1. If the defined statement is in the first place and has **START** specified, the range (*MINVALUE*, **START**) will be automatically used as the first actual partition.
2. The **START END** syntax must comply with the following rules:
   - The value of **START** (if any, same for the following situations) in each **partition_start_end_item** must be smaller than that of **END**.
   - In two adjacent **partition_start_end_item** statements, the value of the first **END** must be equal to that of the second **START**.
   - The value of **EVERY** in each **partition_start_end_item** must be a positive number (in ascending order) and must be smaller than **END** minus **START**.
   - Each partition includes the start value (unless it is *MINVALUE*) and excludes the end value. The format is as follows: [**START**, **END**).
   - Partitions created by the same **partition_start_end_item** belong to the same tablespace.
   - If **partition_name** is a name prefix of a partition, the length must not exceed 57 bytes. If there are more than 57 bytes, the prefix will be automatically truncated.
   - When creating or modifying a partitioned table, ensure that the total number of partitions in the table does not exceed the maximum value **1048575**.
3. In statements for creating partitioned tables, **START END** and **LESS THAN** cannot be used together.
4. The **START END** syntax in a partitioned table creation SQL statement will be replaced by the **VALUES LESS THAN** syntax when **gs_dump** is executed.

- **INTERVAL ('interval_expr') [STORE IN (tablespace_name [, ...])]**

  Defines interval partitioning.

  - **interval_expr**: interval for automatically creating partitions, for example, 1 day or 1 month.
  - **STORE IN (tablespace_name [, ...])**: Specifies the list of tablespaces for storing automatically created partitions. If this parameter is specified, the automatically created partitions are cyclically selected from the tablespace list. Otherwise, the default tablespace of the partition table is used.

NOTICE

Column-store tables do not support interval partitioning.

- **PARTITION BY LIST(partition_key)**

  Create a list partition. **partition_key** is the name of the partition key.

  - For **partition_key**, the list partitioning policy supports only one column of partition keys.

  - For the clause syntax VALUES (list_values), if **list_values** contains the key values of the corresponding partition, it is recommended that the number of key values of each partition be less than or equal to 64.

  Partition keys support the following data types: INT1, INT2, INT4, INT8, NUMERIC, VARCHAR(*n*), CHAR, BPCHAR, NVARCHAR, NVARCHAR2, TIMESTAMP[(*p*)] [WITHOUT TIME ZONE], TIMESTAMP[(*p*)] [WITH TIME ZONE], and DATE. The number of partitions cannot exceed 1048575.

- **PARTITION BY HASH(partition_key)**

  Create a hash partition. **partition_key** is the name of the partition key.

  For **partition_key**, the hash partitioning policy supports only one column of partition keys.

  Partition keys support the following data types: INT1, INT2, INT4, INT8, NUMERIC, VARCHAR(*n*), CHAR, BPCHAR, TEXT, NVARCHAR, NVARCHAR2, TIMESTAMP[(*p*)] [WITHOUT TIME ZONE], TIMESTAMP[(*p*)] [WITH TIME ZONE], and DATE. The number of partitions cannot exceed 1048575.

- **{ ENABLE | DISABLE } ROW MOVEMENT**

  Sets row movement.

  If the tuple value is updated on the partition key during the **UPDATE** action, the partition where the tuple is located is altered. Setting this parameter enables error messages to be reported or movement of the tuple between partitions.

  Value range:

  - **ENABLE** (default value): Row movement is enabled.
  - **DISABLE**: Row movement is disabled.

  If the row movement is enabled, an error may be reported when update and delete operations are performed concurrently. The causes are as follows:

  If the GaussDB Astore engine is used, the old data is marked as deleted under the update and delete operations. If the row movement is enabled, the cross-partition update occurs when the partition key is updated, the GaussDB Astore engine marks the old data in the old partition as deleted and adds a data to the new partition. As a result, the new data cannot be found by querying the old data.

  If data in the same row is concurrently operated, the cross-partition and non-cross-partition data results have different behaviors in the following three concurrency scenarios: update and update concurrency, delete and delete concurrency, as well as update and delete concurrency.

  a. For non-cross-partition data, no error is reported for the second operation after the first operation is performed.

     If the first operation is update, the latest data can be found and operated after the second operation is performed.

     If the first operation is delete, the second operation is terminated if the current data is deleted and the latest data cannot be found.

  b. For the cross-partition data result, an error is reported for the second operation after the first operation is performed.

     If the first operation is update, the second operation cannot find the latest data because the new data is in the new partition. Therefore, the second operation fails and an error is reported.

If the first operation is delete, performing the second operation can find that the current data is deleted and the latest data cannot be found, but cannot determine whether the operation of deleting the old data is update or delete. If the operation is update, an error is reported. If the operation is delete, the second operation is terminated. To ensure the data correctness, an error is reported.

If the update and update concurrency, and update and delete concurrency are performed, the error can be solved only when the operations are performed serially. If the delete and delete concurrency are performed, the error can be solved by disabling the row movement.

- **NOT NULL**

  The column is not allowed to contain null values. **ENABLE** can be omitted.

- **NULL**

  Specifies that the column is allowed to contain null values. This is the default setting.

  This clause is only provided for compatibility with non-standard SQL databases. It is not recommended.

- **CHECK (condition) [NO INHERIT]**

  Specifies an expression producing a Boolean result where the INSERT or UPDATE operation of new or updated rows can succeed only when the expression result is **TRUE** or **UNKNOWN**; otherwise, an error is thrown and the database is not altered.

  A check constraint specified as a column constraint should reference only the column's values, while an expression in a table constraint can reference multiple columns.

  A constraint marked with **NO INHERIT** will not propagate to child tables.

  ENABLE can be omitted.

- **DEFAULT default_expr**

  Assigns a default data value to a column. The value can be any variable-free expressions. (Subqueries and cross-references to other columns in the current table are not allowed.) The data type of the default expression must match the data type of the column.

  The default expression will be used in any insert operation that does not specify a value for the column. If there is no default value for a column, then the default value is null.

- GENERATED ALWAYS AS (generation_expr) STORED

  This clause creates a column as a generated column. The value of the generated column is calculated by **generation_expr** when data is written (inserted or updated). **STORED** indicates that the value of the generated column is stored as a common column.

NOTE

- The generation expression cannot refer to data other than the current row in any way. The generation expression cannot reference other generation columns or system columns. The generation expression cannot return a result set. No subquery, aggregate function, or window function can be used. The function called by the generation expression can only be an immutable function.
- Default values cannot be specified for generated columns.
- The generated column cannot be used as a part of the partition key.
- Do not specify the generated column and the CASCADE, SET NULL, and SET DEFAULT actions of the ON UPDATE constraint at the same time. Do not specify the generated column and the SET NULL, and SET DEFAULT actions of the ON DELETE constraint at the same time.
- The method of modifying and deleting generated columns is the same as that of common columns. Delete the common column that the generated column depends on. The generated column is automatically deleted. The type of the column on which the generated column depends cannot be changed.
- The generated column cannot be directly written. In the INSERT or UPDATE statement, values cannot be specified for generated columns, but the keyword DEFAULT can be specified.
- The permission control for generated columns is the same as that for common columns.
- Columns cannot be generated for column-store tables and MOTs. In foreign tables, only **postgres_fdw** supports generated columns.

- **UNIQUE index_parameters**

  UNIQUE (column_name [, ...]) index_parameters

  Specifies that a group of one or more columns of a table can contain only unique values.

  For the purpose of a unique constraint, null is not considered equal.
- **PRIMARY KEY index_parameters**

  PRIMARY KEY (column_name [, ...]) index_parameters

  Specifies that a column or columns of a table can contain only unique (non-duplicate) and non-null values.

  Only one primary key can be specified for a table.
- **DEFERRABLE | NOT DEFERRABLE**

  Controls whether the constraint can be deferred. A constraint that is not deferrable will be checked immediately after every command. Checking of constraints that are deferrable can be postponed until the end of the transaction using the **SET CONSTRAINTS** command. **NOT DEFERRABLE** is the default value. Currently, only UNIQUE constraints, primary key constraints, and foreign key constraints accept this clause. All the other constraints are not deferrable.
- **INITIALLY IMMEDIATE | INITIALLY DEFERRED**

  If a constraint is deferrable, this clause specifies the default time to check the constraint.
  - If the constraint is **INITIALLY IMMEDIATE** (default value), it is checked after each statement.
  - If the constraint is **INITIALLY DEFERRED**, it is checked only at the end of the transaction.

The constraint check time can be altered using the **SET CONSTRAINTS** statement.

- **USING INDEX TABLESPACE tablespace_name**

  Allows selection of the tablespace in which the index associated with a **UNIQUE** or **PRIMARY KEY** constraint will be created. If not specified, **default_tablespace** is consulted, or the default tablespace in the database if **default_tablespace** is empty.

Examples

- Example 1: Create a range-partitioned table **tpcds.web_returns_p1**. The table has eight partitions and their partition keys are of the integer type. The ranges of the partitions are: wr_returned_date_sk < 2450815, 2450815 ≤ wr_returned_date_sk < 2451179, 2451179 ≤ wr_returned_date_sk < 2451544, 2451544 ≤ wr_returned_date_sk < 2451910, 2451910 ≤ wr_returned_date_sk < 2452275, 2452275 ≤ wr_returned_date_sk < 2452640, 2452640 ≤ wr_returned_date_sk < 2453005, and wr_returned_date_sk ≥ 2453005.

```
-- Create the tpcds.web_returns table.
openGauss=# CREATE TABLE tpcds.web_returns
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)                   ,
    W_WAREHOUSE_SQ_FT         INTEGER                       ,
    W_STREET_NUMBER           CHAR(10)                      ,
    W_STREET_NAME             VARCHAR(60)                   ,
    W_STREET_TYPE             CHAR(15)                      ,
    W_SUITE_NUMBER            CHAR(10)                      ,
    W_CITY                    VARCHAR(60)                   ,
    W_COUNTY                  VARCHAR(30)                   ,
    W_STATE                   CHAR(2)                       ,
    W_ZIP                     CHAR(10)                      ,
    W_COUNTRY                 VARCHAR(20)                   ,
    W_GMT_OFFSET              DECIMAL(5,2)
);
-- Create a range-partitioned table tpcds.web_returns_p1.
openGauss=# CREATE TABLE tpcds.web_returns_p1
(
    WR_RETURNED_DATE_SK       INTEGER                       ,
    WR_RETURNED_TIME_SK       INTEGER                       ,
    WR_ITEM_SK                INTEGER               NOT NULL,
    WR_REFUNDED_CUSTOMER_SK   INTEGER                       ,
    WR_REFUNDED_CDEMO_SK      INTEGER                       ,
    WR_REFUNDED_HDEMO_SK      INTEGER                       ,
    WR_REFUNDED_ADDR_SK       INTEGER                       ,
    WR_RETURNING_CUSTOMER_SK  INTEGER                       ,
    WR_RETURNING_CDEMO_SK     INTEGER                       ,
    WR_RETURNING_HDEMO_SK     INTEGER                       ,
    WR_RETURNING_ADDR_SK      INTEGER                       ,
    WR_WEB_PAGE_SK            INTEGER                       ,
    WR_REASON_SK              INTEGER                       ,
    WR_ORDER_NUMBER           BIGINT                NOT NULL,
    WR_RETURN_QUANTITY        INTEGER                       ,
    WR_RETURN_AMT             DECIMAL(7,2)                  ,
    WR_RETURN_TAX             DECIMAL(7,2)                  ,
    WR_RETURN_AMT_INC_TAX     DECIMAL(7,2)                  ,
    WR_FEE                    DECIMAL(7,2)                  ,
    WR_RETURN_SHIP_COST       DECIMAL(7,2)                  ,
    WR_REFUNDED_CASH          DECIMAL(7,2)                  ,
    WR_REVERSED_CHARGE        DECIMAL(7,2)                  ,
    WR_ACCOUNT_CREDIT         DECIMAL(7,2)                  ,
    WR_NET_LOSS               DECIMAL(7,2)
)
WITH (ORIENTATION = COLUMN,COMPRESSION=MIDDLE)
```

```
PARTITION BY RANGE(WR_RETURNED_DATE_SK)
(
        PARTITION P1 VALUES LESS THAN(2450815),
        PARTITION P2 VALUES LESS THAN(2451179),
        PARTITION P3 VALUES LESS THAN(2451544),
        PARTITION P4 VALUES LESS THAN(2451910),
        PARTITION P5 VALUES LESS THAN(2452275),
        PARTITION P6 VALUES LESS THAN(2452640),
        PARTITION P7 VALUES LESS THAN(2453005),
        PARTITION P8 VALUES LESS THAN(MAXVALUE)
);

-- Import data from the example data table.
openGauss=# INSERT INTO tpcds.web_returns_p1 SELECT * FROM tpcds.web_returns;

-- Delete the P8 partition.
openGauss=# ALTER TABLE tpcds.web_returns_p1 DROP PARTITION P8;

-- Add a partition WR_RETURNED_DATE_SK with values ranging from 2453005 to 2453105.
openGauss=# ALTER TABLE tpcds.web_returns_p1 ADD PARTITION P8 VALUES LESS THAN (2453105);

-- Add a partition WR_RETURNED_DATE_SK with values ranging from 2453105 to MAXVALUE.
openGauss=# ALTER TABLE tpcds.web_returns_p1 ADD PARTITION P9 VALUES LESS THAN
(MAXVALUE);

-- Delete the P8 partition.
openGauss=# ALTER TABLE tpcds.web_returns_p1 DROP PARTITION FOR (2453005);

-- Rename the P7 partition to P10.
openGauss=# ALTER TABLE tpcds.web_returns_p1 RENAME PARTITION P7 TO P10;

-- Rename the P6 partition to P11.
openGauss=# ALTER TABLE tpcds.web_returns_p1 RENAME PARTITION FOR (2452639) TO P11;

-- Query the number of rows in the P10 partition.
openGauss=# SELECT count(*) FROM tpcds.web_returns_p1 PARTITION (P10);
 count
--------
 0
(1 row)

-- Query the number of rows in the P1 partition.
openGauss=# SELECT COUNT(*) FROM tpcds.web_returns_p1 PARTITION FOR (2450815);
 count
--------
 0
(1 row)
```

- Example 2: Create a range-partitioned table **tpcds.web_returns_p2**. The table has eight partitions and their partition keys are of the integer type. The upper limit of the eighth partition is *MAXVALUE*.

  The ranges of the partitions are: wr_returned_date_sk < 2450815, 2450815 ≤ wr_returned_date_sk < 2451179, 2451179 ≤ wr_returned_date_sk < 2451544, 2451544 ≤ wr_returned_date_sk < 2451910, 2451910 ≤ wr_returned_date_sk < 2452275, 2452275 ≤ wr_returned_date_sk < 2452640, 2452640 ≤ wr_returned_date_sk < 2453005, and wr_returned_date_sk ≥ 2453005.

  The tablespace of the **tpcds.web_returns_p2** partitioned table is **example1**. Partitions **P1** to **P7** have no specified tablespaces, and use the **example1** tablespace of the **tpcds.web_returns_p2** partitioned table. The tablespace of the **P8** partitioned table is **example2**.

  Assume that the following data directories of the database nodes are empty directories for which user **dwsadmin** has the read and write permissions: **/pg_location/mount1/path1**, **/pg_location/mount2/path2**, **/pg_location/mount3/path3**, and **/pg_location/mount4/path4**.

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

openGauss=# CREATE TABLE tpcds.web_returns_p2
(
    WR_RETURNED_DATE_SK       INTEGER                       ,
    WR_RETURNED_TIME_SK       INTEGER                       ,
    WR_ITEM_SK                INTEGER               NOT NULL,
    WR_REFUNDED_CUSTOMER_SK   INTEGER                       ,
    WR_REFUNDED_CDEMO_SK      INTEGER                       ,
    WR_REFUNDED_HDEMO_SK      INTEGER                       ,
    WR_REFUNDED_ADDR_SK       INTEGER                       ,
    WR_RETURNING_CUSTOMER_SK  INTEGER                       ,
    WR_RETURNING_CDEMO_SK     INTEGER                       ,
    WR_RETURNING_HDEMO_SK     INTEGER                       ,
    WR_RETURNING_ADDR_SK      INTEGER                       ,
    WR_WEB_PAGE_SK            INTEGER                       ,
    WR_REASON_SK              INTEGER                       ,
    WR_ORDER_NUMBER           BIGINT                NOT NULL,
    WR_RETURN_QUANTITY        INTEGER                       ,
    WR_RETURN_AMT             DECIMAL(7,2)                  ,
    WR_RETURN_TAX             DECIMAL(7,2)                  ,
    WR_RETURN_AMT_INC_TAX     DECIMAL(7,2)                  ,
    WR_FEE                    DECIMAL(7,2)                  ,
    WR_RETURN_SHIP_COST       DECIMAL(7,2)                  ,
    WR_REFUNDED_CASH          DECIMAL(7,2)                  ,
    WR_REVERSED_CHARGE        DECIMAL(7,2)                  ,
    WR_ACCOUNT_CREDIT         DECIMAL(7,2)                  ,
    WR_NET_LOSS               DECIMAL(7,2)
)
TABLESPACE example1
PARTITION BY RANGE(WR_RETURNED_DATE_SK)
(
        PARTITION P1 VALUES LESS THAN(2450815),
        PARTITION P2 VALUES LESS THAN(2451179),
        PARTITION P3 VALUES LESS THAN(2451544),
        PARTITION P4 VALUES LESS THAN(2451910),
        PARTITION P5 VALUES LESS THAN(2452275),
        PARTITION P6 VALUES LESS THAN(2452640),
        PARTITION P7 VALUES LESS THAN(2453005),
        PARTITION P8 VALUES LESS THAN(MAXVALUE) TABLESPACE example2
)
ENABLE ROW MOVEMENT;

-- Create a partitioned table using LIKE.
openGauss=# CREATE TABLE tpcds.web_returns_p3 (LIKE tpcds.web_returns_p2 INCLUDING
PARTITION);

-- Change the tablespace of the P1 partition to example2.
openGauss=# ALTER TABLE tpcds.web_returns_p2 MOVE PARTITION P1 TABLESPACE example2;

-- Change the tablespace of the P2 partition to example3.
openGauss=# ALTER TABLE tpcds.web_returns_p2 MOVE PARTITION P2 TABLESPACE example3;

-- Split the P8 partition at 2453010.
openGauss=# ALTER TABLE tpcds.web_returns_p2 SPLIT PARTITION P8 AT (2453010) INTO
(
        PARTITION P9,
        PARTITION P10
);

-- Merge the P6 and P7 partitions into one.
openGauss=# ALTER TABLE tpcds.web_returns_p2 MERGE PARTITIONS P6, P7 INTO PARTITION P8;

-- Modify the migration attribute of the partitioned table.
openGauss=# ALTER TABLE tpcds.web_returns_p2 DISABLE ROW MOVEMENT;
-- Delete tables and tablespaces.
```

```
openGauss=# DROP TABLE tpcds.web_returns_p1;
openGauss=# DROP TABLE tpcds.web_returns_p2;
openGauss=# DROP TABLE tpcds.web_returns_p3;
openGauss=# DROP TABLESPACE example1;
openGauss=# DROP TABLESPACE example2;
openGauss=# DROP TABLESPACE example3;
openGauss=# DROP TABLESPACE example4;
```

- Example 3: Use **START END** to create and modify a range-partitioned table.

  Assume that **/home/omm/startend_tbs1**, **/home/omm/startend_tbs2**, **/home/omm/startend_tbs3**, and **/home/omm/startend_tbs4** are empty directories for which user **omm** has the read and write permissions.

```
-- Create tablespaces.
openGauss=# CREATE TABLESPACE startend_tbs1 LOCATION '/home/omm/startend_tbs1';
openGauss=# CREATE TABLESPACE startend_tbs2 LOCATION '/home/omm/startend_tbs2';
openGauss=# CREATE TABLESPACE startend_tbs3 LOCATION '/home/omm/startend_tbs3';
openGauss=# CREATE TABLESPACE startend_tbs4 LOCATION '/home/omm/startend_tbs4';

-- Create a temporary schema.
openGauss=# CREATE SCHEMA tpcds;
openGauss=# SET CURRENT_SCHEMA TO tpcds;

-- Create a partitioned table with the partition key of the integer type.
openGauss=# CREATE TABLE tpcds.startend_pt (c1 INT, c2 INT)
TABLESPACE startend_tbs1
PARTITION BY RANGE (c2) (
    PARTITION p1 START(1) END(1000) EVERY(200) TABLESPACE startend_tbs2,
    PARTITION p2 END(2000),
    PARTITION p3 START(2000) END(2500) TABLESPACE startend_tbs3,
    PARTITION p4 START(2500),
    PARTITION p5 START(3000) END(5000) EVERY(1000) TABLESPACE startend_tbs4
)
ENABLE ROW MOVEMENT;

-- View the information of the partitioned table.
openGauss=# SELECT relname, boundaries, spcname FROM pg_partition p JOIN pg_tablespace t ON
p.reltablespace=t.oid and p.parentid='tpcds.startend_pt'::regclass ORDER BY 1;
   relname   | boundaries |    spcname
-------------+------------+---------------
 p1_0        | {1}        | startend_tbs2
 p1_1        | {201}      | startend_tbs2
 p1_2        | {401}      | startend_tbs2
 p1_3        | {601}      | startend_tbs2
 p1_4        | {801}      | startend_tbs2
 p1_5        | {1000}     | startend_tbs2
 p2          | {2000}     | startend_tbs1
 p3          | {2500}     | startend_tbs3
 p4          | {3000}     | startend_tbs1
 p5_1        | {4000}     | startend_tbs4
 p5_2        | {5000}     | startend_tbs4
 startend_pt |            | startend_tbs1
(12 rows)

-- Import data and check the data volume in a partition.
openGauss=# INSERT INTO tpcds.startend_pt VALUES (GENERATE_SERIES(0, 4999),
GENERATE_SERIES(0, 4999));
openGauss=# SELECT COUNT(*) FROM tpcds.startend_pt PARTITION FOR (0);
 count
-------
     1
(1 row)

openGauss=# SELECT COUNT(*) FROM tpcds.startend_pt PARTITION (p3);
 count
-------
   500
(1 row)
```

```
-- Add partitions [5000, 5300), [5300, 5600), [5600, 5900), and [5900, 6000).
openGauss=# ALTER TABLE tpcds.startend_pt ADD PARTITION p6 START(5000) END(6000)
EVERY(300) TABLESPACE startend_tbs4;

-- Add the partition p7, specified by MAXVALUE.
openGauss=# ALTER TABLE tpcds.startend_pt ADD PARTITION p7 END(MAXVALUE);

-- Rename the partition p7 to p8.
openGauss=# ALTER TABLE tpcds.startend_pt RENAME PARTITION p7 TO p8;

-- Delete the partition p8.
openGauss=# ALTER TABLE tpcds.startend_pt DROP PARTITION p8;

-- Rename the partition where 5950 is located to p71.
openGauss=# ALTER TABLE tpcds.startend_pt RENAME PARTITION FOR(5950) TO p71;

-- Split the partition [4000, 5000) where 4500 is located.
openGauss=# ALTER TABLE tpcds.startend_pt SPLIT PARTITION FOR(4500) INTO(PARTITION q1
START(4000) END(5000) EVERY(250) TABLESPACE startend_tbs3);

-- Change the tablespace of the partition p2 to startend_tbs4.
openGauss=# ALTER TABLE tpcds.startend_pt MOVE PARTITION p2 TABLESPACE startend_tbs4;

-- View the partition status.
openGauss=# SELECT relname, boundaries, spcname FROM pg_partition p JOIN pg_tablespace t ON
p.reltablespace=t.oid and p.parentid='tpcds.startend_pt'::regclass ORDER BY 1;
   relname   | boundaries |    spcname
-------------+------------+---------------
 p1_0        | {1}        | startend_tbs2
 p1_1        | {201}      | startend_tbs2
 p1_2        | {401}      | startend_tbs2
 p1_3        | {601}      | startend_tbs2
 p1_4        | {801}      | startend_tbs2
 p1_5        | {1000}     | startend_tbs2
 p2          | {2000}     | startend_tbs4
 p3          | {2500}     | startend_tbs3
 p4          | {3000}     | startend_tbs1
 p5_1        | {4000}     | startend_tbs4
 p6_1        | {5300}     | startend_tbs4
 p6_2        | {5600}     | startend_tbs4
 p6_3        | {5900}     | startend_tbs4
 p71         | {6000}     | startend_tbs4
 q1_1        | {4250}     | startend_tbs3
 q1_2        | {4500}     | startend_tbs3
 q1_3        | {4750}     | startend_tbs3
 q1_4        | {5000}     | startend_tbs3
 startend_pt |            | startend_tbs1
(19 rows)

-- Delete tables and tablespaces.
openGauss=# DROP SCHEMA tpcds CASCADE;
openGauss=# DROP TABLESPACE startend_tbs1;
openGauss=# DROP TABLESPACE startend_tbs2;
openGauss=# DROP TABLESPACE startend_tbs3;
openGauss=# DROP TABLESPACE startend_tbs4;
```

- Example 4: Create interval partitioned table **sales**. The table initially contains two partitions and the partition key is of the DATE type. Ranges of the two partitions are as follows: **time_id** < '2019-02-01 00:00:00' and '2019-02-01 00:00:00' ≤ **time_id** < '2019-02-02 00:00:00', respectively.

```
-- Create table sales.
openGauss=# CREATE TABLE sales
(prod_id NUMBER(6),
 cust_id NUMBER,
 time_id DATE,
 channel_id CHAR(1),
 promo_id NUMBER(6),
 quantity_sold NUMBER(3),
 amount_sold NUMBER(10,2)
)
PARTITION BY RANGE (time_id)
INTERVAL('1 day')
( PARTITION p1 VALUES LESS THAN ('2019-02-01 00:00:00'),
  PARTITION p2 VALUES LESS THAN ('2019-02-02 00:00:00')
);

-- Insert data into partition p1.
openGauss=# INSERT INTO sales VALUES(1, 12, '2019-01-10 00:00:00', 'a', 1, 1, 1);

-- Insert data into partition p2.
openGauss=# INSERT INTO sales VALUES(1, 12, '2019-02-01 00:00:00', 'a', 1, 1, 1);

-- View the partition information.
openGauss=# SELECT t1.relname, partstrategy, boundaries FROM pg_partition t1, pg_class t2 WHERE
t1.parentid = t2.oid AND t2.relname = 'sales' AND t1.parttype = 'p';
 relname | partstrategy |        boundaries
---------+--------------+-------------------------
 p1      | r            | {"2019-02-01 00:00:00"}
 p2      | r            | {"2019-02-02 00:00:00"}
(2 rows)

-- If the data to be inserted does not match any partition, create a partition and insert the data into
the new partition.
-- The range of the new partition is '2019-02-05 00:00:00' ≤ time_id < '2019-02-06 00:00:00'.
openGauss=# INSERT INTO sales VALUES(1, 12, '2019-02-05 00:00:00', 'a', 1, 1, 1);

-- If the data to be inserted does not match any partition, create a partition and insert the data into
the new partition.
-- The range of the new partition is '2019-02-03 00:00:00' ≤ time_id < '2019-02-04 00:00:00'.
openGauss=# INSERT INTO sales VALUES(1, 12, '2019-02-03 00:00:00', 'a', 1, 1, 1);

-- View the partition information.
openGauss=# SELECT t1.relname, partstrategy, boundaries FROM pg_partition t1, pg_class t2 WHERE
t1.parentid = t2.oid AND t2.relname = 'sales' AND t1.parttype = 'p';
 relname | partstrategy |        boundaries
---------+--------------+-------------------------
 sys_p1  | i            | {"2019-02-06 00:00:00"}
 sys_p2  | i            | {"2019-02-04 00:00:00"}
 p1      | r            | {"2019-02-01 00:00:00"}
 p2      | r            | {"2019-02-02 00:00:00"}
(4 rows)
```

- Example 5: Create list partitioned table **test_list**. The table initially contains four partitions and the partition key is of the INT type. The ranges of the four partitions are 2000, 3000, 4000, and 5000 respectively.

```
-- Create the test_list table.
openGauss=# create table test_list (col1 int, col2 int)
partition by list(col1)
(
partition p1 values (2000),
partition p2 values (3000),
partition p3 values (4000),
partition p4 values (5000)
);

-- Insert data.
openGauss=# INSERT INTO test_list VALUES(2000, 2000);
INSERT 0 1
openGauss=# INSERT INTO test_list VALUES(3000, 3000);
INSERT 0 1

-- View the partition information.
openGauss=# SELECT t1.relname, partstrategy, boundaries FROM pg_partition t1, pg_class t2 WHERE
t1.parentid = t2.oid AND t2.relname = 'test_list' AND t1.parttype = 'p';
 relname | partstrategy | boundaries
---------+--------------+------------
 p1      | l            | {2000}
 p2      | l            | {3000}
 p3      | l            | {4000}
 p4      | l            | {5000}
(4 rows)

-- The inserted data does not match the partition, and an error is reported.
openGauss=# INSERT INTO test_list VALUES(6000, 6000);
ERROR:  inserted partition key does not map to any table partition

-- Add a partition.
openGauss=# alter table test_list add partition p5 values (6000);
ALTER TABLE
openGauss=# SELECT t1.relname, partstrategy, boundaries FROM pg_partition t1, pg_class t2 WHERE
t1.parentid = t2.oid AND t2.relname = 'test_list' AND t1.parttype = 'p';
 relname | partstrategy | boundaries
---------+--------------+------------
 p5      | l            | {6000}
 p4      | l            | {5000}
 p1      | l            | {2000}
 p2      | l            | {3000}
 p3      | l            | {4000}
(5 rows)
openGauss=# INSERT INTO test_list VALUES(6000, 6000);
INSERT 0 1

-- Exchange data between the partitioned table and ordinary table.
openGauss=# create table t1 (col1 int, col2 int);
CREATE TABLE
openGauss=# select * from test_list partition (p1);
 col1 | col2
------+------
 2000 | 2000
(1 row)
openGauss=# alter table test_list exchange partition (p1) with table t1;
ALTER TABLE
openGauss=# select * from test_list partition (p1);
 col1 | col2
------+------
(0 rows)
openGauss=# select * from t1;
 col1 | col2
------+------
 2000 | 2000
(1 row)

-- Truncate the partition.
openGauss=# select * from test_list partition (p2);
 col1 | col2
------+------
 3000 | 3000
(1 row)
openGauss=# alter table test_list truncate partition p2;
ALTER TABLE
openGauss=# select * from test_list partition (p2);
 col1 | col2
------+------
(0 rows)

-- Delete the partition.
openGauss=# alter table test_list drop partition p5;
ALTER TABLE
openGauss=# SELECT t1.relname, partstrategy, boundaries FROM pg_partition t1, pg_class t2 WHERE
t1.parentid = t2.oid AND t2.relname = 'test_list' AND t1.parttype = 'p';
 relname | partstrategy | boundaries
---------+--------------+------------
 p4      | l            | {5000}
 p1      | l            | {2000}
 p2      | l            | {3000}
 p3      | l            | {4000}
(4 rows)
```

```
openGauss=# INSERT INTO test_list VALUES(6000, 6000);
ERROR:  inserted partition key does not map to any table partition

-- Delete the partitioned table.
openGauss=# drop table test_list;
```

- Example 6: Create a hash partitioned table **test_hash**. The table initially contains two partitions and the partition key is of the INT type.

```
-- Create the test_hash table.
openGauss=# create table test_hash (col1 int, col2 int)
partition by hash(col1)
(
partition p1,
partition p2
);

-- Insert data.
openGauss=# INSERT INTO test_hash VALUES(1, 1);
INSERT 0 1
openGauss=# INSERT INTO test_hash VALUES(2, 2);
INSERT 0 1
openGauss=# INSERT INTO test_hash VALUES(3, 3);
INSERT 0 1
openGauss=# INSERT INTO test_hash VALUES(4, 4);
INSERT 0 1

-- View the partition information.
openGauss=# SELECT t1.relname, partstrategy, boundaries FROM pg_partition t1, pg_class t2 WHERE
t1.parentid = t2.oid AND t2.relname = 'test_hash' AND t1.parttype = 'p';
 relname | partstrategy | boundaries
---------+--------------+------------
 p1      | h            | {0}
 p2      | h            | {1}
(2 rows)

-- View the data.
openGauss=# select * from test_hash partition (p1);
 col1 | col2
------+------
    3 |    3
    4 |    4
(2 rows)

openGauss=# select * from test_hash partition (p2);
 col1 | col2
------+------
    1 |    1
    2 |    2
(2 rows)

-- Exchange data between the partitioned table and ordinary table.
openGauss=# create table t1 (col1 int, col2 int);
CREATE TABLE
openGauss=# alter table test_hash exchange partition (p1) with table t1;
ALTER TABLE
openGauss=# select * from test_hash partition (p1);
 col1 | col2
------+------
(0 rows)
openGauss=# select * from t1;
 col1 | col2
------+------
    3 |    3
    4 |    4
(2 rows)

-- Truncate the partition.
openGauss=# alter table test_hash truncate partition p2;
ALTER TABLE
```

```
openGauss=#  select * from test_hash partition (p2);
 col1 | col2
------+------
(0 rows)

-- Delete the partitioned table.
openGauss=# drop table test_hash;
```

Helpful Links

ALTER TABLE PARTITION and DROP TABLE.

11.14.89 CREATE TABLESPACE

Function

CREATE TABLESPACE creates a tablespace in a database.

Precautions

- The system administrator or a user who inherits the gs_role_tablespace permission of the built-in role can create a tablespace.
- Do not run **CREATE TABLESPACE** in a transaction block.
- If executing **CREATE TABLESPACE** fails but the internal directory (or file) has been created, the directory (or file) will remain. You need to manually clear it before creating the tablespace again. If there are residual files of soft links for the tablespace in the data directory, delete the residual files, and then perform O&M operations.
- **CREATE TABLESPACE** cannot be used for two-phase transactions. If it fails on some nodes, the execution cannot be rolled back.
- For details about how to prepare for creating tablespaces, see the description of parameters below.
- You are not advised to use user-defined tablespaces in scenarios such as Huawei Cloud. This is because user-defined tablespaces are usually used with storage media other than the main storage (storage device where the default tablespace is located, such as a disk) to isolate I/O resources that can be used by different services. Storage devices use standard configurations and do not have other available storage media in scenarios such as Huawei Cloud. If the user-defined tablespace is not properly used, the system cannot run stably for a long time and the overall performance is affected. Therefore, you are advised to use the default tablespace.

Syntax

```
CREATE TABLESPACE tablespace_name
    [ OWNER user_name ] [RELATIVE] LOCATION 'directory' [ MAXSIZE 'space_size' ]
    [with_option_clause];
```

The **with_option_clause** syntax for creating a general tablespace is as follows:

```
WITH ( {filesystem= { 'general'| "general" | general} |
    random_page_cost = { 'value ' | value } |
    seq_page_cost = { 'value ' | value }}[,...])
```

Parameter Description

- **tablespace_name**

  Specifies name of a tablespace to be created.

  The tablespace name must be distinct from the name of any existing tablespace in the database and cannot start with "pg", which are reserved for system catalog spaces.

  Value range: a string. It must comply with the identifier naming convention.

- **OWNER user_name**

  Specifies the name of the user who will own the tablespace. If omitted, the default owner is the current user.

  Only system administrators can create tablespaces, but they can use the **OWNER** clause to assign ownership of tablespaces to non-**Sysadmin** administrators.

  Value range: a string. It must be an existing user.

- **RELATIVE**

  Relative path. The **LOCATION** directory is relative to the data directory in each database node.

  Directory hierarchy: the relative path of the database node directory **/pg_location/**

  A relative path contains a maximum of two levels.

- **LOCATION directory**

  Specifies the directory used for the tablespace. The directory must meet the following requirements:

  - The GaussDB system user must have the read and write permissions on the directory, and the directory must be empty. If the directory does not exist, the system automatically creates it.
  - The directory must be an absolute path, and does not contain special characters, such as dollar sign ($) and greater-than sign (>).
  - The directory cannot be specified under the database data directory.
  - The directory must be a local path.

  Value range: a string. It must be a valid directory.

- **MAXSIZE 'space_size'**

  Specifies the maximum value of the tablespace in a single database node.

  Value range: a string consisting of a positive integer and unit. The unit can be KB, MB, GB, TB, or PB currently. The unit of parsed value is KB and cannot exceed the range that can be expressed in 64 bits, which is 1 KB to 9007199254740991 KB.

- **random_page_cost**

  Specifies the cost of randomly reading the page overhead.

  Value range: 0 to 1.79769e+308

  Default value: value of the GUC parameter **random_page_cost**

- **seq_page_cost**

  Specifies the cost of reading the page overhead in specified order.

  Value range: 0 to 1.79769e+308

Default value: value of GUC parameter **seq_page_cost**

Examples

```
-- Create a tablespace.
openGauss=# CREATE TABLESPACE ds_location1 RELATIVE LOCATION 'tablespace/tablespace_1';

-- Create user joe.
openGauss=# CREATE ROLE joe IDENTIFIED BY 'xxxxxxxxx';

-- Create user jay.
openGauss=# CREATE ROLE jay IDENTIFIED BY 'xxxxxxxxx';

-- Create a tablespace and set its owner to user joe.
openGauss=# CREATE TABLESPACE ds_location2 OWNER joe RELATIVE LOCATION 'tablespace/
tablespace_1';

-- Rename the ds_location1 tablespace to ds_location3.
openGauss=# ALTER TABLESPACE ds_location1 RENAME TO ds_location3;

-- Change the owner of the ds_location2 tablespace.
openGauss=# ALTER TABLESPACE ds_location2 OWNER TO jay;

-- Delete the tablespace.
openGauss=# DROP TABLESPACE ds_location2;
openGauss=# DROP TABLESPACE ds_location3;

-- Delete users.
openGauss=# DROP ROLE joe;
openGauss=# DROP ROLE jay;
```

Helpful Links

CREATE DATABASE, **CREATE TABLE**, **CREATE INDEX**, **DROP TABLESPACE**, and **ALTER TABLESPACE**

Suggestions

- create tablespace

  You are not advised to create tablespaces in a transaction.

11.14.90 CREATE TABLE SUBPARTITION

Function

CREATE TABLE SUBPARTITION creates a level-2 partitioned table. A partitioned table is a logical table that is divided into several physical partitions for storage based on a specific plan. A partitioned table is a logical table and does not store data. Data is stored in physical partitions. For a level-2 partitioned table, the top-level node table and level-1 partitioned table are logical tables and do not store data. Only the level-2 partitioned (leaf node) stores data.

The partitioning solution of a level-2 partitioned table is a combination of the partitioning solutions of two level-1 partitions. For details about the partitioning solution of a level-1 partitioned table, see CREATE TABLE PARTITION.

Common combination solutions for level-2 partitioned tables include range-range partitioning, range-list partitioning, range-hash partitioning, list-range partitioning, list-list partitioning, list-hash partitioning, hash-range partitioning, hash-list partitioning, and hash-hash partitioning. Currently, level-2 partitioned tables can only be row-store tables.

Precautions

- A level-2 partitioned table has two partition keys, and each partition key supports only one column. The two partition keys cannot be the same column.
- If the constraint key of the unique constraint and primary key constraint contains all partition keys, a local index is created for the constraints. Otherwise, a global index is created. If a local unique index is created, all partition keys must be included.
- When a level-2 partitioned table is created, if the specified level-2 partition is not displayed under the level-1 partition, a level-2 partition with the same range is automatically created.
- The number of level-2 partitions (leaf nodes) in a level-2 partitioned table cannot exceed 1048575. There is no limit on the number of level-1 partitions, but there must be at least one level-2 partition under a level-1 partition.
- The maximum total number of partitions (including level-1 and level-2 partitions) in a level-2 partitioned table is 1048575. Generally, it is impossible for services to create so many partitions and the memory will be insufficient if so. Create partitions based on the value of **local_syscache_threshold**. The memory allocated to the level-2 partitioned tables can be calculated as follows: total number of partitions x 3/1024, in MB. Theoretically, the memory occupied by the partitions cannot be greater than the value of **local_syscache_threshold**. In addition, some space must be reserved for other functions.
- Level-2 partitioned tables support only row store and do not support column-store and hash bucket.
- Clusters are not supported.
- When specifying a partition for query, for example, **select * from tablename partition/subpartition** (*partitionname*), ensure that the keywords **partition** and **subpartition** are correct. If they are incorrect, no error is reported during the query. In this case, the query is performed based on the table alias.
- Encrypted databases, ledger databases, and row-level security are not supported.
- In the **PARTITION FOR (values)** syntax for level-2 partitioned tables, values can only be constants.
- In the **PARTITION/SUBPARTITION FOR (values)** syntax for level-2 partitioned tables, if data type conversion is required for values, you are advised to use forcible type conversion to prevent the implicit type conversion result from being inconsistent with the expected result.
- Currently, the statement specifying a partition cannot perform global index scan.

Syntax

```
CREATE TABLE [ IF NOT EXISTS ] subpartition_table_name
(
{ column_name data_type [ COLLATE collation ] [ column_constraint [ ... ] ]
| table_constraint
| LIKE source_table [ like_option [...] ] }[, ... ]
)
[ WITH ( {storage_parameter = value} [, ... ] ) ]
[ COMPRESS | NOCOMPRESS ]
[ TABLESPACE tablespace_name ]
PARTITION BY {RANGE | LIST | HASH} (partition_key) SUBPARTITION BY {RANGE | LIST | HASH}
(subpartition_key)
(
  PARTITION partition_name1 [ VALUES LESS THAN (val1) | VALUES (val1[, …]) ] [ TABLESPACE tablespace ]
  [(
      { SUBPARTITION subpartition_name1 [ VALUES LESS THAN (val1_1) | VALUES (val1_1[, …])]
[ TABLESPACE tablespace ] } [, ...]
  )][, ...]
)[ { ENABLE | DISABLE } ROW MOVEMENT ];
```

- Column constraint:
```
[ CONSTRAINT constraint_name ]
{ NOT NULL |
  NULL |
  CHECK ( expression ) |
  DEFAULT default_e xpr |
  GENERATED ALWAYS AS ( generation_expr ) STORED |
  UNIQUE index_parameters |
  PRIMARY KEY index_parameters |
  REFERENCES reftable [ ( refcolumn ) ] [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ]
        [ ON DELETE action ] [ ON UPDATE action ] }
[ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```
- Table constraint:
```
[ CONSTRAINT constraint_name ]
{ CHECK ( expression ) |
  UNIQUE ( column_name [, ... ] ) index_parameters |
  PRIMARY KEY ( column_name [, ... ] ) index_parameters |
  FOREIGN KEY ( column_name [, ... ] ) REFERENCES reftable [ ( refcolumn [, ... ] ) ]
      [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ] [ ON DELETE action ] [ ON UPDATE
action ] }
[ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```
- LIKE options:
```
{ INCLUDING | EXCLUDING } { DEFAULTS | GENERATED | CONSTRAINTS | INDEXES | STORAGE |
COMMENTS | RELOPTIONS| ALL }
```
- Index parameters:
```
[ WITH ( {storage_parameter = value} [, ... ] ) ]
[ USING INDEX TABLESPACE tablespace_name ]
```

Parameter Description

- **IF NOT EXISTS**

  Does not throw an error if a relationship with the same name existed. A notice is issued in this case.
- **subpartition_table_name**

  Specifies the name of a level-2 partitioned table.

  Value range: a string. It must comply with the identifier naming convention.
- **column_name**

  Specifies the name of a column to be created in the new table.

  Value range: a string. It must comply with the identifier naming convention.
- **data_type**

  Specifies the data type of the column.
- **COLLATE collation**

  Assigns a collation to the column (which must be of a collatable data type). If no collation is specified, the default collation is used. You can run the **select ***

from pg_collation; command to query collation rules from the **pg_collation** system catalog. The default collation rule is the row starting with **default** in the query result.

- **CONSTRAINT constraint_name**

  Specifies the name of a column or table constraint. The optional constraint clauses specify constraints that new or updated rows must satisfy for an INSERT or UPDATE operation to succeed.

  There are two ways to define constraints:

  – A column constraint is defined as part of a column definition, and it is bound to a particular column.

  – A table constraint is not bound to a particular column but can apply to more than one column.

- **LIKE source_table [like_option ...]**

  Level-2 partitioned tables do not support this function.

- **WITH (storage_parameter [= value] [, ...])**

  Specifies an optional storage parameter for a table or an index. Optional parameters are as follows:

  – FILLFACTOR

    The fill factor of a table is a percentage from 10 to 100. **100** (complete filling) is the default value. When a smaller fill factor is specified, INSERT operations fill table pages only to the indicated percentage. The remaining space on each page is reserved for updating rows on that page. This gives UPDATE a chance to place the updated copy of a row on the same page, which is more efficient than placing it on a different page. For a table whose entries are never updated, setting the fill factor to **100** (complete filling) is the best choice, but in heavily updated tables a smaller fill factor would be appropriate. The parameter has no meaning for column-store tables.

    Value range: 10–100

  – ORIENTATION

    Determines the data storage mode of the table.

    Value range:

    ▪ **COLUMN**: The data will be stored in columns.

    ▪ **ROW** (default value): The data will be stored in rows.

    NOTICE

    ORIENTATION cannot be modified.

  – STORAGE_TYPE

    Specifies the storage engine type. This parameter cannot be modified once it is set.

    Value range:

    ▪ **USTORE** indicates that tables support the inplace-update storage engine. Note that the **track_counts** and **track_activities** parameters must be enabled when the Ustore table is used. Otherwise, space expansion may occur.

  - **ASTORE** indicates that tables support the append-only storage engine.

  Default value:

  If no table is specified, data is stored in append-only mode by default.

- COMPRESSION
  - Value range: **LOW**, **MIDDLE**, **HIGH**, **YES**, and **NO** for column-store tables, with compression level increasing in ascending order. The default value is **LOW**.
  - Row-store tables do not support compression.
- MAX_BATCHROW

  Specifies the maximum number of records in a storage unit during data loading. The parameter is only valid for column-store tables.

  Value range: 10000 to 60000. The default value is **60000**.
- PARTIAL_CLUSTER_ROWS

  Specifies the number of records to be partially clustered for storage during data loading. The parameter is only valid for column-store tables.

  Value range: greater than or equal to **MAX_BATCHROW**. You are advised to set this parameter to an integer multiple of **MAX_BATCHROW**.
- DELTAROW_THRESHOLD

  A reserved parameter. The parameter is only valid for column-store tables.

  Value range: 0 to 9999
- segment

  The data is stored in segment-page mode. This parameter supports only row-store tables. Column-store tables, temporary tables, and unlogged tables are not supported. The Ustore storage engine is not supported.

  Value range: **on** and **off**

  Default value: **off**

- **COMPRESS / NOCOMPRESS**

  Specifies keyword COMPRESS during the creation of a table, so that the compression feature is triggered in case of BULK INSERT operations. If this feature is enabled, a scan is performed for all tuple data within the page to generate a dictionary and then the tuple data is compressed and stored. If **NOCOMPRESS** is specified, the table is not compressed. Row-store tables do not support compression.

  Default value: **NOCOMPRESS**, that is, tuple data is not compressed before storage.
- **TABLESPACE tablespace_name**

  Specifies that the new table will be created in the **tablespace_name** tablespace. If the tablespace is not specified, the default tablespace is used.
- **PARTITION BY {RANGE | LIST | HASH} (partition_key)**

- For **partition_key**, the partitioning policy supports only one column of partition keys.
- The data types supported by the partition key are the same as those supported by the level-1 partitioned table.

- **SUBPARTITION BY {RANGE | LIST | HASH} (subpartition_key)**
  - For **subpartition_key**, the partitioning policy supports only one column of partition keys.
  - The data types supported by the partition key are the same as those supported by the level-1 partitioned table.
- **{ ENABLE | DISABLE } ROW MOVEMENT**

  Specifies whether to enable row movement.

  If the tuple value is updated on the partition key during the UPDATE operation, the partition where the tuple is located is altered. Setting this parameter enables error messages to be reported or movement of the tuple between partitions.

  Value range:

  - **ENABLE** (default value): Row movement is enabled.
  - **DISABLE**: Row movement is disabled.

  If the row movement is enabled, an error may be reported when update and delete operations are performed concurrently. The causes are as follows:

  If the GaussDB Astore engine is used, the old data is marked as deleted data under the update and delete operations. If the row movement is enabled, the cross-partition update occurs when the partition key is updated, the GaussDB Astore engine marks the old data in the old partition as deleted data and adds a piece of data to the new partition. As a result, the new data cannot be found by querying the old data.

  If data in the same row is concurrently operated, the cross-partition and non-cross-partition data results have different behaviors in the following three concurrency scenarios: update and update concurrency, delete and delete concurrency, update and delete concurrency.

  a. For non-cross-partition data, no error is reported for the second operation after the first operation is performed.

     If the first operation is update, the latest data can be found and operated after the second operation is performed.

     If the first operation is delete, the second operation is terminated if the current data is deleted and the latest data cannot be found.

  b. For the cross-partition data result, an error is reported for the second operation after the first operation is performed.

     If the first operation is update, the second operation cannot find the latest data because the new data is in the new partition. Therefore, the second operation fails and an error is reported.

     If the first operation is delete, performing the second operation can find that the current data is deleted and the latest data cannot be found, but cannot determine whether the operation of deleting the old data is update or delete. If the operation is update, an error is reported. If the operation is delete, the second operation is terminated. To ensure the data correctness, an error is reported.

If the update and update concurrency, and update and delete concurrency are performed, the error can be solved only when the operations are performed serially. If the delete and delete concurrency are performed, the error can be solved by disabling the row movement.

- **NOT NULL**

  The column is not allowed to contain null values. **ENABLE** can be omitted.

- **NULL**

  Indicates that the column is allowed to contain **NULL** values. This is the default setting.

  This clause is only provided for compatibility with non-standard SQL databases. It is not recommended.

- **CHECK (condition) [NO INHERIT]**

  Specifies an expression producing a Boolean result where the INSERT or UPDATE operation of new or updated rows can succeed only when the expression result is **TRUE** or **UNKNOWN**; otherwise, an error is thrown and the database is not altered.

  A check constraint specified as a column constraint should reference only the column's values, while an expression in a table constraint can reference multiple columns.

  A constraint marked with **NO INHERIT** will not propagate to child tables.

  ENABLE can be omitted.

- **DEFAULT default_expr**

  Assigns a default data value to a column. The value can be any variable-free expressions. (Subqueries and cross-references to other columns in the current table are not allowed.) The data type of the default expression must match that of the column.

  The default expression will be used in any INSERT operation that does not specify a value for the column. If there is no default value for a column, then the default value is **NULL**.

- **GENERATED ALWAYS AS (generation_expr) STORED**

  This clause creates a column as a generated column. The value of the generated column is calculated by **generation_expr** when data is written (inserted or updated). **STORED** indicates that the value of the generated column is stored as a common column.

NOTE

- The generation expression cannot refer to data other than the current row in any way. The generation expression cannot reference other generation columns or system columns. The generation expression cannot return a result set. No subquery, aggregate function, or window function can be used. The function called by the generation expression can only be an immutable function.
- Default values cannot be specified for generated columns.
- The generated column cannot be used as a part of the partition key.
- Do not specify the generated column and the CASCADE, SET NULL, and SET DEFAULT actions of the ON UPDATE constraint at the same time. Do not specify the generated column and the SET NULL, and SET DEFAULT actions of the ON DELETE constraint at the same time.
- The method of modifying and deleting generated columns is the same as that of common columns. Delete the common column that the generated column depends on. The generated column is automatically deleted. The type of the column on which the generated column depends cannot be changed.
- The generated column cannot be directly written. In the INSERT or UPDATE statement, values cannot be specified for generated columns, but the keyword DEFAULT can be specified.
- The permission control for generated columns is the same as that for common columns.
- Columns cannot be generated for column-store tables and MOTs. In foreign tables, only **postgres_fdw** supports generated columns.

- **UNIQUE index_parameters**

  UNIQUE (column_name [, ...]) index_parameters

  Specifies that a group of one or more columns of a table can contain only unique values.

  For the purpose of a unique constraint, null is not considered equal.

- **PRIMARY KEY index_parameters**

  PRIMARY KEY (column_name [, ...]) index_parameters

  Specifies that a column or columns of a table can contain only unique (non-duplicate) and non-null values.

  Only one primary key can be specified for a table.

- **DEFERRABLE | NOT DEFERRABLE**

  They determine whether the constraint is deferrable. A constraint that is not deferrable will be checked immediately after every command. Checking of constraints that are deferrable can be postponed until the end of the transaction using the **SET CONSTRAINTS** command. **NOT DEFERRABLE** is the default value. Currently, only UNIQUE constraints, primary key constraints, and foreign key constraints accept this clause. All the other constraints are not deferrable.

- **INITIALLY IMMEDIATE | INITIALLY DEFERRED**

  If a constraint is deferrable, this clause specifies the default time to check the constraint.

  – If the constraint is **INITIALLY IMMEDIATE** (default value), it is checked after each statement.
  – If the constraint is **INITIALLY DEFERRED**, it is checked only at the end of the transaction.

The constraint check time can be altered using the **SET CONSTRAINTS** statement.

- **USING INDEX TABLESPACE tablespace_name**

  Allows selection of the tablespace in which the index associated with a **UNIQUE** or **PRIMARY KEY** constraint will be created. If not specified, the index is created in **default_tablespace**. If **default_tablespace** is empty, the default tablespace of the database is used.

Examples

- Example 1: Create level-2 partitioned tables of various combination types.

```
CREATE TABLE list_list
(
    month_code VARCHAR2 ( 30 ) NOT NULL ,
    dept_code  VARCHAR2 ( 30 ) NOT NULL ,
    user_no    VARCHAR2 ( 30 ) NOT NULL ,
    sales_amt  int
)
PARTITION BY LIST (month_code) SUBPARTITION BY LIST (dept_code)
(
  PARTITION p_201901 VALUES ( '201902' )
  (
    SUBPARTITION p_201901_a VALUES ( '1' ),
    SUBPARTITION p_201901_b VALUES ( '2' )
  ),
  PARTITION p_201902 VALUES ( '201903' )
  (
    SUBPARTITION p_201902_a VALUES ( '1' ),
    SUBPARTITION p_201902_b VALUES ( '2' )
  )
);
insert into list_list values('201902', '1', '1', 1);
insert into list_list values('201902', '2', '1', 1);
insert into list_list values('201902', '1', '1', 1);
insert into list_list values('201903', '2', '1', 1);
insert into list_list values('201903', '1', '1', 1);
insert into list_list values('201903', '2', '1', 1);
select * from list_list;
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201903     | 2         | 1       |         1
 201903     | 2         | 1       |         1
 201903     | 1         | 1       |         1
 201902     | 2         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
(6 rows)

drop table list_list;
CREATE TABLE list_hash
(
    month_code VARCHAR2 ( 30 ) NOT NULL ,
    dept_code  VARCHAR2 ( 30 ) NOT NULL ,
    user_no    VARCHAR2 ( 30 ) NOT NULL ,
    sales_amt  int
)
PARTITION BY LIST (month_code) SUBPARTITION BY HASH (dept_code)
(
  PARTITION p_201901 VALUES ( '201902' )
  (
    SUBPARTITION p_201901_a,
    SUBPARTITION p_201901_b
  ),
  PARTITION p_201902 VALUES ( '201903' )
  (
    SUBPARTITION p_201902_a,
    SUBPARTITION p_201902_b
  )
);
insert into list_hash values('201902', '1', '1', 1);
insert into list_hash values('201902', '2', '1', 1);
insert into list_hash values('201902', '3', '1', 1);
insert into list_hash values('201903', '4', '1', 1);
insert into list_hash values('201903', '5', '1', 1);
insert into list_hash values('201903', '6', '1', 1);
select * from list_hash;
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201903     | 4         | 1       |         1
 201903     | 5         | 1       |         1
 201903     | 6         | 1       |         1
 201902     | 2         | 1       |         1
 201902     | 3         | 1       |         1
 201902     | 1         | 1       |         1
(6 rows)

drop table list_hash;
CREATE TABLE list_range
(
    month_code VARCHAR2 ( 30 ) NOT NULL ,
    dept_code  VARCHAR2 ( 30 ) NOT NULL ,
    user_no    VARCHAR2 ( 30 ) NOT NULL ,
    sales_amt  int
)
PARTITION BY LIST (month_code) SUBPARTITION BY RANGE (dept_code)
(
  PARTITION p_201901 VALUES ( '201902' )
  (
    SUBPARTITION p_201901_a values less than ('4'),
    SUBPARTITION p_201901_b values less than ('6')
  ),
  PARTITION p_201902 VALUES ( '201903' )
  (
    SUBPARTITION p_201902_a values less than ('3'),
    SUBPARTITION p_201902_b values less than ('6')
  )
);
insert into list_range values('201902', '1', '1', 1);
insert into list_range values('201902', '2', '1', 1);
insert into list_range values('201902', '3', '1', 1);
insert into list_range values('201903', '4', '1', 1);
insert into list_range values('201903', '5', '1', 1);
insert into list_range values('201903', '6', '1', 1);
ERROR:  inserted partition key does not map to any table partition
select * from list_range;
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201903     | 4         | 1       |         1
 201903     | 5         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 2         | 1       |         1
 201902     | 3         | 1       |         1
(5 rows)

drop table list_range;
CREATE TABLE range_list
(
    month_code VARCHAR2 ( 30 ) NOT NULL ,
    dept_code  VARCHAR2 ( 30 ) NOT NULL ,
    user_no    VARCHAR2 ( 30 ) NOT NULL ,
    sales_amt  int
)
PARTITION BY RANGE (month_code) SUBPARTITION BY LIST (dept_code)
(
  PARTITION p_201901 VALUES LESS THAN( '201903' )
  (
    SUBPARTITION p_201901_a values ('1'),
    SUBPARTITION p_201901_b values ('2')
  ),
  PARTITION p_201902 VALUES LESS THAN( '201904' )
  (
    SUBPARTITION p_201902_a values ('1'),
    SUBPARTITION p_201902_b values ('2')
  )
);
insert into range_list values('201902', '1', '1', 1);
insert into range_list values('201902', '2', '1', 1);
insert into range_list values('201902', '1', '1', 1);
insert into range_list values('201903', '2', '1', 1);
insert into range_list values('201903', '1', '1', 1);
insert into range_list values('201903', '2', '1', 1);
select * from range_list;
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902     | 2         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
 201903     | 2         | 1       |         1
 201903     | 2         | 1       |         1
 201903     | 1         | 1       |         1
(6 rows)

drop table range_list;
CREATE TABLE range_hash
(
    month_code VARCHAR2 ( 30 ) NOT NULL ,
    dept_code  VARCHAR2 ( 30 ) NOT NULL ,
    user_no    VARCHAR2 ( 30 ) NOT NULL ,
    sales_amt  int
)
PARTITION BY RANGE (month_code) SUBPARTITION BY HASH (dept_code)
(
  PARTITION p_201901 VALUES LESS THAN( '201903' )
  (
    SUBPARTITION p_201901_a,
    SUBPARTITION p_201901_b
  ),
  PARTITION p_201902 VALUES LESS THAN( '201904' )
  (
    SUBPARTITION p_201902_a,
    SUBPARTITION p_201902_b
  )
);
insert into range_hash values('201902', '1', '1', 1);
insert into range_hash values('201902', '2', '1', 1);
insert into range_hash values('201902', '1', '1', 1);
insert into range_hash values('201903', '2', '1', 1);
insert into range_hash values('201903', '1', '1', 1);
insert into range_hash values('201903', '2', '1', 1);
select * from range_hash;
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902     | 2         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
 201903     | 2         | 1       |         1
 201903     | 2         | 1       |         1
 201903     | 1         | 1       |         1
(6 rows)

drop table range_hash;
CREATE TABLE range_range
(
    month_code VARCHAR2 ( 30 ) NOT NULL ,
    dept_code  VARCHAR2 ( 30 ) NOT NULL ,
    user_no    VARCHAR2 ( 30 ) NOT NULL ,
    sales_amt  int
)
PARTITION BY RANGE (month_code) SUBPARTITION BY RANGE (dept_code)
(
  PARTITION p_201901 VALUES LESS THAN( '201903' )
  (
    SUBPARTITION p_201901_a VALUES LESS THAN( '2' ),
    SUBPARTITION p_201901_b VALUES LESS THAN( '3' )
  ),
  PARTITION p_201902 VALUES LESS THAN( '201904' )
  (
    SUBPARTITION p_201902_a VALUES LESS THAN( '2' ),
    SUBPARTITION p_201902_b VALUES LESS THAN( '3' )
  )
);
insert into range_range values('201902', '1', '1', 1);
insert into range_range values('201902', '2', '1', 1);
insert into range_range values('201902', '1', '1', 1);
insert into range_range values('201903', '2', '1', 1);
insert into range_range values('201903', '1', '1', 1);
insert into range_range values('201903', '2', '1', 1);
select * from range_range;
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 2         | 1       |         1
 201903     | 1         | 1       |         1
 201903     | 2         | 1       |         1
 201903     | 2         | 1       |         1
(6 rows)

drop table range_range;
CREATE TABLE hash_list
(
    month_code VARCHAR2 ( 30 ) NOT NULL ,
    dept_code  VARCHAR2 ( 30 ) NOT NULL ,
    user_no    VARCHAR2 ( 30 ) NOT NULL ,
    sales_amt  int
)
PARTITION BY hash (month_code) SUBPARTITION BY LIST (dept_code)
(
  PARTITION p_201901
  (
    SUBPARTITION p_201901_a VALUES ( '1' ),
    SUBPARTITION p_201901_b VALUES ( '2' )
  ),
  PARTITION p_201902
  (
    SUBPARTITION p_201902_a VALUES ( '1' ),
    SUBPARTITION p_201902_b VALUES ( '2' )
  )
);
insert into hash_list values('201901', '1', '1', 1);
insert into hash_list values('201901', '2', '1', 1);
insert into hash_list values('201901', '1', '1', 1);
insert into hash_list values('201903', '2', '1', 1);
insert into hash_list values('201903', '1', '1', 1);
insert into hash_list values('201903', '2', '1', 1);
select * from hash_list;
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201903     | 2         | 1       |         1
 201903     | 2         | 1       |         1
 201903     | 1         | 1       |         1
 201901     | 2         | 1       |         1
 201901     | 1         | 1       |         1
 201901     | 1         | 1       |         1
(6 rows)

drop table hash_list;
CREATE TABLE hash_hash
(
    month_code VARCHAR2 ( 30 ) NOT NULL ,
    dept_code  VARCHAR2 ( 30 ) NOT NULL ,
    user_no    VARCHAR2 ( 30 ) NOT NULL ,
    sales_amt  int
)
PARTITION BY hash (month_code) SUBPARTITION BY hash (dept_code)
(
  PARTITION p_201901
  (
    SUBPARTITION p_201901_a,
    SUBPARTITION p_201901_b
  ),
  PARTITION p_201902
  (
    SUBPARTITION p_201902_a,
    SUBPARTITION p_201902_b
  )
);
insert into hash_hash values('201901', '1', '1', 1);
insert into hash_hash values('201901', '2', '1', 1);
insert into hash_hash values('201901', '1', '1', 1);
insert into hash_hash values('201903', '2', '1', 1);
insert into hash_hash values('201903', '1', '1', 1);
insert into hash_hash values('201903', '2', '1', 1);
select * from hash_hash;
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201903     | 2         | 1       |         1
 201903     | 2         | 1       |         1
 201903     | 1         | 1       |         1
 201901     | 2         | 1       |         1
 201901     | 1         | 1       |         1
 201901     | 1         | 1       |         1
(6 rows)

drop table hash_hash;
CREATE TABLE hash_range
(
    month_code VARCHAR2 ( 30 ) NOT NULL ,
    dept_code  VARCHAR2 ( 30 ) NOT NULL ,
    user_no    VARCHAR2 ( 30 ) NOT NULL ,
    sales_amt  int
)
PARTITION BY hash (month_code) SUBPARTITION BY range (dept_code)
(
  PARTITION p_201901
  (
    SUBPARTITION p_201901_a VALUES LESS THAN ( '2' ),
    SUBPARTITION p_201901_b VALUES LESS THAN ( '3' )
  ),
  PARTITION p_201902
  (
    SUBPARTITION p_201902_a VALUES LESS THAN ( '2' ),
    SUBPARTITION p_201902_b VALUES LESS THAN ( '3' )
  )
);
insert into hash_range values('201901', '1', '1', 1);
insert into hash_range values('201901', '2', '1', 1);
insert into hash_range values('201901', '1', '1', 1);
insert into hash_range values('201903', '2', '1', 1);
insert into hash_range values('201903', '1', '1', 1);
insert into hash_range values('201903', '2', '1', 1);
select * from hash_range;
```

```
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201903     | 1         | 1       |         1
 201903     | 2         | 1       |         1
 201903     | 2         | 1       |         1
 201901     | 1         | 1       |         1
 201901     | 1         | 1       |         1
 201901     | 2         | 1       |         1
(6 rows)
```

- Example 2: Specify partitions in a level-2 partitioned table using DML.

```
CREATE TABLE range_list
(
    month_code VARCHAR2 ( 30 ) NOT NULL ,
    dept_code  VARCHAR2 ( 30 ) NOT NULL ,
    user_no    VARCHAR2 ( 30 ) NOT NULL ,
    sales_amt  int
)
PARTITION BY RANGE (month_code) SUBPARTITION BY LIST (dept_code)
(
  PARTITION p_201901 VALUES LESS THAN( '201903' )
  (
    SUBPARTITION p_201901_a values ('1'),
    SUBPARTITION p_201901_b values ('2')
  ),
  PARTITION p_201902 VALUES LESS THAN( '201910' )
  (
    SUBPARTITION p_201902_a values ('1'),
    SUBPARTITION p_201902_b values ('2')
  )
);
-- Insert data to a specified level-1 partition.
insert into range_list partition (p_201901) values('201902', '1', '1', 1);
-- The actual partition is inconsistent with the specified partition. An error is reported.
insert into range_list partition (p_201902) values('201902', '1', '1', 1);
ERROR:  inserted partition key does not map to the table partition
DETAIL:  N/A.
-- Insert data to a specified level-2 partition.
insert into range_list subpartition (p_201901_a) values('201902', '1', '1', 1);
-- The actual partition is inconsistent with the specified partition. An error is reported.
insert into range_list subpartition (p_201901_b) values('201902', '1', '1', 1);
ERROR:  inserted subpartition key does not map to the table subpartition
DETAIL:  N/A.
insert into range_list partition for ('201902') values('201902', '1', '1', 1);
insert into range_list subpartition for ('201902','1') values('201902', '1', '1', 1);

-- Query data in a specified partition.
select * from range_list partition (p_201901);
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
(4 rows)

select * from range_list subpartition (p_201901_a);
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
(4 rows)

select * from range_list partition for ('201902');
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
(4 rows)

select * from range_list subpartition for ('201902','1');
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
(4 rows)

-- Update data in a specified partition.
update range_list partition (p_201901) set user_no = '2';
select * from range_list;
select *from range_list; month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902     | 1         | 2       |         1
 201902     | 1         | 2       |         1
 201902     | 1         | 2       |         1
 201902     | 1         | 2       |         1
(4 rows)
update range_list subpartition (p_201901_a) set user_no = '3';
select * from range_list;
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902     | 1         | 3       |         1
 201902     | 1         | 3       |         1
 201902     | 1         | 3       |         1
 201902     | 1         | 3       |         1
(4 rows)
update range_list partition for ('201902') set user_no = '4';
select * from range_list;
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902     | 1         | 4       |         1
 201902     | 1         | 4       |         1
 201902     | 1         | 4       |         1
 201902     | 1         | 4       |         1
(4 rows)
update range_list subpartition for ('201902','2') set user_no = '5';
openGauss=# select *from range_list;
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902     | 1         | 4       |         1
 201902     | 1         | 4       |         1
 201902     | 1         | 4       |         1
 201902     | 1         | 4       |         1
(4 rows)
select * from range_list;

-- Delete data from a specified partition.
delete from range_list partition (p_201901);
DELETE 4
delete from range_list partition for ('201903');
DELETE 0
delete from range_list subpartition (p_201901_a);
DELETE 0
delete from range_list subpartition for ('201903','2');
DELETE 0

-- Insert data into a specified partition.
insert into range_list partition (p_201901)  values('201902', '1', '1', 1)  ON DUPLICATE KEY UPDATE
sales_amt = 5;
insert into range_list subpartition (p_201901_a)  values('201902', '1', '1', 1)  ON DUPLICATE KEY
UPDATE sales_amt = 10;
insert into range_list partition for ('201902')  values('201902', '1', '1', 1)  ON DUPLICATE KEY UPDATE
sales_amt = 30;
```

```
insert into range_list subpartition for ('201902','1')  values('201902', '1', '1', 1)  ON DUPLICATE KEY
UPDATE sales_amt = 40;
select * from range_list;
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
(4 rows)

-- Merge data into a specified partition.
CREATE TABLE newrange_list
(
    month_code VARCHAR2 ( 30 ) NOT NULL ,
    dept_code  VARCHAR2 ( 30 ) NOT NULL ,
    user_no    VARCHAR2 ( 30 ) NOT NULL ,
    sales_amt  int
)
PARTITION BY RANGE (month_code) SUBPARTITION BY LIST (dept_code)
(
  PARTITION p_201901 VALUES LESS THAN( '201903' )
  (
    SUBPARTITION p_201901_a values ('1'),
    SUBPARTITION p_201901_b values ('2')
  ),
  PARTITION p_201902 VALUES LESS THAN( '201910' )
  (
    SUBPARTITION p_201902_a values ('1'),
    SUBPARTITION p_201902_b values ('2')
  )
);
insert into newrange_list values('201902', '1', '1', 1);
insert into newrange_list values('201903', '1', '1', 2);

MERGE INTO range_list partition (p_201901) p
USING newrange_list partition (p_201901) np
ON p.month_code= np.month_code
WHEN MATCHED THEN
  UPDATE SET dept_code = np.dept_code, user_no = np.user_no, sales_amt = np.sales_amt
WHEN NOT MATCHED THEN
  INSERT VALUES (np.month_code, np.dept_code, np.user_no, np.sales_amt);

select * from range_list;
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
(4 rows)

MERGE INTO range_list partition for ('201901') p
USING newrange_list partition for ('201901') np
ON p.month_code= np.month_code
WHEN MATCHED THEN
  UPDATE SET dept_code = np.dept_code, user_no = np.user_no, sales_amt = np.sales_amt
WHEN NOT MATCHED THEN
  INSERT VALUES (np.month_code, np.dept_code, np.user_no, np.sales_amt);

select * from range_list;
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
(4 rows)
```

```
MERGE INTO range_list subpartition (p_201901_a) p
USING newrange_list subpartition (p_201901_a) np
ON p.month_code= np.month_code
WHEN MATCHED THEN
  UPDATE SET dept_code = np.dept_code, user_no = np.user_no, sales_amt = np.sales_amt
WHEN NOT MATCHED THEN
  INSERT VALUES (np.month_code, np.dept_code, np.user_no, np.sales_amt);

select * from range_list;
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
(4 rows)

MERGE INTO range_list subpartition for ('201901', '1') p
USING newrange_list subpartition for ('201901', '1') np
ON p.month_code= np.month_code
WHEN MATCHED THEN
  UPDATE SET dept_code = np.dept_code, user_no = np.user_no, sales_amt = np.sales_amt
WHEN NOT MATCHED THEN
  INSERT VALUES (np.month_code, np.dept_code, np.user_no, np.sales_amt);

select * from range_list;
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
(4 rows)
```

- Example 3: Truncate a level-2 partitioned table.

```
CREATE TABLE list_list
(
    month_code VARCHAR2 ( 30 ) NOT NULL ,
    dept_code  VARCHAR2 ( 30 ) NOT NULL ,
    user_no    VARCHAR2 ( 30 ) NOT NULL ,
    sales_amt  int
)
PARTITION BY LIST (month_code) SUBPARTITION BY LIST (dept_code)
(
  PARTITION p_201901 VALUES ( '201902' )
  (
    SUBPARTITION p_201901_a VALUES ( '1' ),
    SUBPARTITION p_201901_b VALUES ( default )
  ),
  PARTITION p_201902 VALUES ( '201903' )
  (
    SUBPARTITION p_201902_a VALUES ( '1' ),
    SUBPARTITION p_201902_b VALUES ( '2' )
  )
);
insert into list_list values('201902', '1', '1', 1);
insert into list_list values('201902', '2', '1', 1);
insert into list_list values('201902', '1', '1', 1);
insert into list_list values('201903', '2', '1', 1);
insert into list_list values('201903', '1', '1', 1);
insert into list_list values('201903', '2', '1', 1);
select * from list_list;
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201903     | 2         | 1       |         1
 201903     | 2         | 1       |         1
 201903     | 1         | 1       |         1
 201902     | 2         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
```

```
(6 rows)

select * from list_list partition (p_201901);
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902     | 2         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
(3 rows)

alter table list_list truncate partition p_201901;
select * from list_list partition (p_201901);
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
(0 rows)

select * from list_list partition (p_201902);
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201903     | 2         | 1       |         1
 201903     | 2         | 1       |         1
 201903     | 1         | 1       |         1
(3 rows)

alter table list_list truncate partition p_201902;
select * from list_list partition (p_201902);
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
(0 rows)

select * from list_list;
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
(0 rows)

insert into list_list values('201902', '1', '1', 1);
insert into list_list values('201902', '2', '1', 1);
insert into list_list values('201902', '1', '1', 1);
insert into list_list values('201903', '2', '1', 1);
insert into list_list values('201903', '1', '1', 1);
insert into list_list values('201903', '2', '1', 1);
select * from list_list subpartition (p_201901_a);
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
(2 rows)

alter table list_list truncate subpartition p_201901_a;
select * from list_list subpartition (p_201901_a);
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
(0 rows)

select * from list_list subpartition (p_201901_b);
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902     | 2         | 1       |         1
(1 row)

alter table list_list truncate subpartition p_201901_b;
select * from list_list subpartition (p_201901_b);
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
(0 rows)

select * from list_list subpartition (p_201902_a);
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
```

```
 201903     | 1         | 1       |         1
(1 row)

alter table list_list truncate subpartition p_201902_a;
select * from list_list subpartition (p_201902_a);
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
(0 rows)

select * from list_list subpartition (p_201902_b);
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201903     | 2         | 1       |         1
 201903     | 2         | 1       |         1
(2 rows)

alter table list_list truncate subpartition p_201902_b;
select * from list_list subpartition (p_201902_b);
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
(0 rows)

select * from list_list;
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
(0 rows)

drop table list_list;
```

- Example 4: Split a level-2 partitioned table.

```
CREATE TABLE list_list
(
    month_code VARCHAR2 ( 30 ) NOT NULL ,
    dept_code  VARCHAR2 ( 30 ) NOT NULL ,
    user_no    VARCHAR2 ( 30 ) NOT NULL ,
    sales_amt  int
)
PARTITION BY LIST (month_code) SUBPARTITION BY LIST (dept_code)
(
  PARTITION p_201901 VALUES ( '201902' )
  (
    SUBPARTITION p_201901_a VALUES ( '1' ),
    SUBPARTITION p_201901_b VALUES ( default )
  ),
  PARTITION p_201902 VALUES ( '201903' )
  (
    SUBPARTITION p_201902_a VALUES ( '1' ),
    SUBPARTITION p_201902_b VALUES ( default )
  )
);
insert into list_list values('201902', '1', '1', 1);
insert into list_list values('201902', '2', '1', 1);
insert into list_list values('201902', '1', '1', 1);
insert into list_list values('201903', '2', '1', 1);
insert into list_list values('201903', '1', '1', 1);
insert into list_list values('201903', '2', '1', 1);
select * from list_list;
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201903     | 2         | 1       |         1
 201903     | 2         | 1       |         1
 201903     | 1         | 1       |         1
 201902     | 2         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
(6 rows)

select * from list_list subpartition (p_201901_a);
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
```

```
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
(2 rows)

select * from list_list subpartition (p_201901_b);
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902     | 2         | 1       |         1
(1 row)

alter table list_list split subpartition p_201901_b values (2) into
(
 subpartition p_201901_b,
 subpartition p_201901_c
);
select * from list_list subpartition (p_201901_a);
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
(2 rows)

select * from list_list subpartition (p_201901_b);
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902     | 2         | 1       |         1
(1 row)

select * from list_list subpartition (p_201901_c);
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
(0 rows)

select * from list_list partition (p_201901);
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201902     | 2         | 1       |         1
 201902     | 1         | 1       |         1
 201902     | 1         | 1       |         1
(3 rows)

select * from list_list subpartition (p_201902_a);
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201903     | 1         | 1       |         1
(1 row)

select * from list_list subpartition (p_201902_b);
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201903     | 2         | 1       |         1
 201903     | 2         | 1       |         1
(2 rows)

alter table list_list split subpartition p_201902_b values (3) into
(
 subpartition p_201902_b,
 subpartition p_201902_c
);
select * from list_list subpartition (p_201902_a);
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201903     | 1         | 1       |         1
(1 row)

select * from list_list subpartition (p_201902_b);
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
(0 rows)
```

```
select * from list_list subpartition (p_201902_c);
 month_code | dept_code | user_no | sales_amt
------------+-----------+---------+-----------
 201903     | 2         | 1       |         1
 201903     | 2         | 1       |         1
(2 rows)

drop table list_list;
```

11.14.91 CREATE TEXT SEARCH CONFIGURATION

Function

CREATE TEXT SEARCH CONFIGURATION creates a text search configuration. A text search configuration specifies a text search parser that can divide a string into tokens, plus dictionaries that can be used to determine which tokens are of interest for searching.

Precautions

- If only the parser is specified, the new text search configuration initially has no mapping from token types to dictionaries, and therefore will ignore all words. Subsequently, **ALTER TEXT SEARCH CONFIGURATION** must be used to create mapping to make the configuration useful. If **COPY** is specified, the parser, mapping and parameters of the text search configuration is copied automatically.
- If the schema name is given, the text search configuration will be created in the specified schema. Otherwise, the configuration will be created in the current schema.
- The user who defines a text search configuration becomes its owner.
- **PARSER** and **COPY** options are mutually exclusive, because when an existing configuration is copied, its parser selection is copied too.
- If only the parser is specified, the new text search configuration initially has no mapping from token types to dictionaries, and therefore will ignore all words.

Syntax

```
CREATE TEXT SEARCH CONFIGURATION name
    ( PARSER = parser_name | COPY = source_config )
    [ WITH ( {configuration_option = value} [, ...] )];
```

Parameter Description

- **name**

  Specifies the name of the text search configuration to be created. The name can be schema-qualified.
- **parser_name**

  Specifies the name of the text search parser to use for this configuration.
- **source_config**

  Specifies the name of an existing text search configuration to copy.
- **configuration_option**

Specifies parameters for the text search configuration, particularly for the parser executed by **parser_name** or contained by **source_config**.

Value range: The default and **ngram** parsers are supported. The parser of default type has no corresponding **configuration_option**. Table 11-119 lists **configuration_option** for **ngram** parsers.

Table 11-119 Configuration parameters for **ngram** parsers

Parser	Parameter	Description	Value Range
ngram	gram_size	Length of word segmentation	Integer, 1 to 4 Default value: **2**
	punctuation_ignore	Whether to ignore punctuations	● **true** (default value): Ignore punctuations. ● **false**: Do not ignore punctuations.
	grapsymbol_ignore	Whether to ignore graphical characters	● **true**: Ignore graphical characters. ● **false** (default value): Do not ignore graphical characters.

Examples

```
-- Create a text search configuration.
openGauss=# CREATE TEXT SEARCH CONFIGURATION ngram2 (parser=ngram) WITH (gram_size = 2,
grapsymbol_ignore = false);

-- Create a text search configuration.
openGauss=# CREATE TEXT SEARCH CONFIGURATION ngram3 (copy=ngram2) WITH (gram_size = 2,
grapsymbol_ignore = false);

-- Add type mapping.
openGauss=# ALTER TEXT SEARCH CONFIGURATION ngram2 ADD MAPPING FOR multisymbol WITH
simple;

-- Create user joe.
openGauss=# CREATE USER joe IDENTIFIED BY 'xxxxxxxxx';

-- Change the owner of the text search configuration.
openGauss=# ALTER TEXT SEARCH CONFIGURATION ngram2 OWNER TO joe;

-- Change the schema of the text search configuration.
openGauss=# ALTER TEXT SEARCH CONFIGURATION ngram2 SET SCHEMA joe;

-- Rename the text search configuration.
openGauss=# ALTER TEXT SEARCH CONFIGURATION joe.ngram2 RENAME TO ngram_2;

-- Delete the type mapping.
openGauss=# ALTER TEXT SEARCH CONFIGURATION joe.ngram_2 DROP MAPPING IF EXISTS FOR
multisymbol;

-- Delete the text search configuration.
openGauss=# DROP TEXT SEARCH CONFIGURATION joe.ngram_2;
openGauss=# DROP TEXT SEARCH CONFIGURATION ngram3;
```

```
-- Delete the schema and user joe.
openGauss=# DROP SCHEMA IF EXISTS joe CASCADE;
openGauss=# DROP ROLE IF EXISTS joe;
```

Helpful Links

ALTER TEXT SEARCH CONFIGURATION and DROP TEXT SEARCH CONFIGURATION

11.14.92 CREATE TEXT SEARCH DICTIONARY

Function

CREATE TEXT SEARCH DICTIONARY creates a full-text retrieval dictionary. A dictionary is used to identify and process particular words during full-text retrieval.

Dictionaries are created by using predefined templates (defined in the PG_TS_TEMPLATE system catalog). Five types of dictionaries can be created, **Simple**, **Ispell**, **Synonym**, **Thesaurus**, and **Snowball**. These dictionaries are used to handle different types of tasks.

Precautions

- A user with the **SYSADMIN** permission can create a dictionary. Then, the user automatically becomes the owner of the dictionary.
- A dictionary cannot be created in **pg_temp** mode.
- After a dictionary is created or modified, any modification to the customized dictionary definition file will not affect the dictionary in the database. To make such modifications take effect in the dictionary in the database, run the **ALTER** statement to update the definition file of the dictionary.

Syntax

```
CREATE TEXT SEARCH DICTIONARY name (
    TEMPLATE = template
    [, option = value [, ... ]]
);
```

Parameter Description

- **name**

  Specifies the name of a dictionary to be created. (If you do not specify a schema name, the dictionary will be created in the current schema.)

  Value range: a string, which complies with the identifier naming convention. A value can contain a maximum of 63 characters.
- **template**

  Specifies a template name.

  Value range: templates (**Simple**, **Synonym**, **Thesaurus**, **Ispell**, and **Snowball**) defined in the PG_TS_TEMPLATE system catalog
- **option**

  Specifies a parameter name. Each type of dictionaries has a template containing their custom parameters. Parameters function in a way irrelevant to their setting sequence.

- Parameters for a **Simple** dictionary
  - **STOPWORDS**

    Specifies the name of a file listing stop words. The default file name extension is .stop. In the file, each line defines a stop word. Dictionaries will ignore blank lines and spaces in the file and convert stop-word phrases into lowercase.
  - **ACCEPT**

    Specifies whether to accept a non-stop word as recognized. Default value: **true**

    If **ACCEPT=true** is set for a **Simple** dictionary, no token will be passed to subsequent dictionaries. In this case, you are advised to place the **Simple** dictionary at the end of the dictionary list. If **ACCEPT=false** is set, you are advised to place the **Simple** dictionary before at least one dictionary in the list.
  - **FILEPATH**

    Specifies the directory for storing dictionary files. The directory can be a local directory or an OBS directory. (The OBS directory can be specified only in security mode. You can add the securitymode option during startup to enter the security mode.) The local directory format is **file://***absolute_path*. The OBS directory format is **obs://***bucket_name***/path accesskey=ak secretkey=sk region=rg**. The default value is the directory where predefined dictionary files are located. If any of the **FILEPATH** and **STOPWORDS** parameters is specified, the other one must also be specified.
- Parameters for a **Synonym** dictionary
  - **SYNONYM**

    Specifies the name of the definition file for a **Synonym** dictionary. The default file name extension is .syn.

    The file is a list of synonyms. Each line is in the format of *token synonym*, that is, token and its synonym separated by a space.
  - **CASESENSITIVE**

    Specifies whether tokens and their synonyms are case sensitive. The default value is **false**, indicating that tokens and synonyms in dictionary files will be converted into lowercase. If this parameter is set to **true**, they will not be converted into lowercase.
  - **FILEPATH**

    Specifies the directory for storing **Synonym** dictionary files. The directory can be a local directory or an OBS directory. (The OBS directory can be specified only in security mode. You can add the **securitymode** option during startup to enter the security mode.) The local directory format is **file://***absolute_path*. The OBS directory format is **obs://***bucket_name***/path accesskey=ak secretkey=sk region=rg**. The default value is the directory where predefined dictionary files are located.
- Parameters for a **Thesaurus** dictionary

- **DICTFILE**

  Specifies the name of a dictionary definition file. The default file name extension is .ths.

  The file is a list of synonyms. Each line is in the format of *sample words* **:** *indexed words*. The colon (:) is used as a separator between a phrase and its substitute word. If multiple sample words are matched, the TZ selects the longest one.

- **DICTIONARY**

  Specifies the name of a subdictionary used for word normalization. This parameter is mandatory and only one subdictionary name can be specified. The specified subdictionary must exist. It is used to identify and normalize input text before phrase matching.

  If an input word cannot be recognized by the subdictionary, an error will be reported. In this case, remove the word or update the subdictionary to make the word recognizable. In addition, an asterisk (*) can be placed at the beginning of an indexed word to skip the application of a subdictionary on it, but all sample words must be recognizable by the subdictionary.

  If the sample words defined in the dictionary file contain stop words defined in the subdictionary, use question marks (?) to replace them. Assume that **a** and **the** are stop words defined in the subdictionary.

  ```
  ? one ? two : swsw
  ```

  a one the two and **the one a two** will be matched and output as **swsw**.

- **FILEPATH**

  Specifies the directory for storing dictionary definition files. The directory can be a local directory or an OBS directory. (The OBS directory can be specified only in security mode. You can add the securitymode option during startup to enter the security mode.) The local directory format is **file://***absolute_path*. The OBS directory format is **obs://***bucket_name***/path accesskey=ak secretkey=sk region=rg**. The default value is the directory where predefined dictionary files are located.

– Parameters for an **Ispell** dictionary

- **DICTFILE**

  Specifies the name of a dictionary definition file. The default file name extension is .dict.

- **AFFFILE**

  Specifies the name of an affix file. The default file name extension is .affix.

- **STOPWORDS**

  Specifies the name of a file listing stop words. The default file name extension is .stop. The file content format is the same as that of the file for a **Simple** dictionary.

- **FILEPATH**

Specifies the directory for storing dictionary files. The directory can be a local directory or an OBS directory. (The OBS directory can be specified only in security mode. You can add the **securitymode** option during startup to enter the security mode.) The local directory format is **file://***absolute_path*. The OBS directory format is **obs://***bucket_name***/path accesskey=ak secretkey=sk region=rg**. The default value is the directory where predefined dictionary files are located.

– Parameters for a **Snowball** dictionary

■ **LANGUAGE**

Specifies the name of a language whose stemming algorithm will be used. According to spelling rules in the language, the algorithm normalizes the variants of an input word into a basic word or a stem.

■ **STOPWORDS**

Specifies the name of a file listing stop words. The default file name extension is .stop. The file content format is the same as that of the file for a **Simple** dictionary.

■ **FILEPATH**

Specifies the directory for storing dictionary definition files. You can specify a local directory or an OBS directory. (The OBS directory can be specified only in security mode. You can enter the security mode by adding the **securitymode** option during startup.) The local directory format is **file://***absolute_path*. The OBS directory format is **obs://***bucket_name***/path accesskey=ak secretkey=sk region=rg**. The default value is the directory where predefined dictionary files are located. If any of the **FILEPATH** and **STOPWORDS** parameters is specified, the other one must also be specified.

NOTE

The name of a dictionary definition file can contain only lowercase letters, digits, and underscores (_).

- **value**

  Specifies a parameter value. If the value is not an identifier or a number, enclose it with single quotation marks (''). You can also enclose identifiers and numbers.

Examples

See examples in **Configuration Examples**.

Helpful Links

ALTER TEXT SEARCH DICTIONARY and **CREATE TEXT SEARCH DICTIONARY**

11.14.93 CREATE TRIGGER

Function

CREATE TRIGGER creates a trigger. The trigger will be associated with the specified table or view, and will execute the specified function operations are performed.

Precautions

- Currently, triggers can be created only on ordinary row-store tables, instead of on column-store tables, temporary tables, or unlogged tables.
- If multiple triggers of the same kind are defined for the same event, they will be fired in alphabetical order by name.
- Triggers are usually used for data association and synchronization between multiple tables. SQL execution performance is greatly affected. Therefore, you are advised not to use this statement when a large amount of data needs to be synchronized and performance requirements are high.

Syntax

```
CREATE [ CONSTRAINT ] TRIGGER trigger_name { BEFORE | AFTER | INSTEAD OF } { event [ OR ... ] }
    ON table_name
    [ FROM referenced_table_name ]
    { NOT DEFERRABLE | [ DEFERRABLE ] { INITIALLY IMMEDIATE | INITIALLY DEFERRED } }
    [ FOR [ EACH ] { ROW | STATEMENT } ]
    [ WHEN ( condition ) ]
    EXECUTE PROCEDURE function_name ( arguments );
```

Events include:

```
    INSERT
    UPDATE [ OF column_name [, ... ] ]
    DELETE
    TRUNCATE
```

Parameter Description

- **CONSTRAINT**

  (Optional) Creates a constraint trigger. That is, the trigger is used as a constraint. This is the same as a regular trigger except that the timing of the trigger firing can be adjusted using **SET CONSTRAINTS**. Constraint triggers must be **AFTER ROW** triggers.

- **trigger_name**

  Specifies the name of the trigger to be created. This must be distinct from the name of any other trigger for the same table. The name cannot be schema-qualified — the trigger inherits the schema of its table. For a constraint trigger, this is also the name to use when modifying the trigger's behavior using **SET CONSTRAINTS**.

  Value range: a string, which complies with the identifier naming convention and contains a maximum of 63 characters.

- **BEFORE**

  Specifies that the function is called before the event.

- **AFTER**

  Specifies that the function is called after the event. A constraint trigger can only be specified as **AFTER**.
- **INSTEAD OF**

  Specifies that the function is called instead of the event.
- **event**

  Specifies the event that will fire the trigger. Values are **INSERT**, **UPDATE**, **DELETE**, and **TRUNCATE**. Multiple events can be specified using **OR**.

  For **UPDATE** events, it is possible to specify a list of columns using this syntax:

  ```
  UPDATE OF column_name1 [, column_name2 ... ]
  ```

  The trigger will only fire if at least one of the listed columns is mentioned as a target of the **UPDATE** statement. **INSTEAD OF UPDATE** events do not support lists of columns. If the column specified by **UPDATE OF** contains a generated column, the trigger is also fired when the column on which the generated column depends is the target column of the **UPDATE** statement.
- **table_name**

  Specifies the name of the table for which the trigger is created.

  Value range: name of an existing table in the database
- **referenced_table_name**

  Specifies the name of another table referenced by the constraint. This option is used for foreign-key constraints. It can only be specified for constraint triggers.

  Value range: name of an existing table in the database
- **DEFERRABLE | NOT DEFERRABLE**

  Specifies the start time of the trigger. It can only be specified for constraint triggers. They determine whether the constraint is deferrable.

  For details, see **CREATE TABLE**.
- **INITIALLY IMMEDIATE | INITIALLY DEFERRED**

  If the constraint is deferrable, the two clauses specify the default time to check the constraint. It can only be specified for constraint triggers.

  For details, see **CREATE TABLE**.
- **FOR EACH ROW | FOR EACH STATEMENT**

  Specifies the frequency of firing the trigger.

  - **FOR EACH ROW** indicates that the trigger should be fired once for every row affected by the trigger event.
  - **FOR EACH STATEMENT** indicates that the trigger should be fired just once per SQL statement.

  If neither is specified, the default is **FOR EACH STATEMENT**. Constraint triggers can only be marked as **FOR EACH ROW**.
- **condition**

  Specifies whether the trigger function will actually be executed. If **WHEN** is specified, the function will be called only when **condition** returns **true**.

  In **FOR EACH ROW** triggers, the **WHEN** condition can refer to columns of the old and/or new row values by writing **OLD.***column name* or **NEW.***column* *name* respectively. In addition, **INSERT** triggers cannot refer to **OLD**, and **DELETE** triggers cannot refer to **NEW**.

INSTEAD OF triggers do not support **WHEN** conditions.

Currently, **WHEN** expressions cannot contain subqueries.

Note that for constraint triggers, evaluation of the **WHEN** condition is not deferred, but occurs immediately after the row update operation is performed. If the condition does not evaluate to **true**, then the trigger is not queued for deferred execution.

- **function_name**

  Specifies a user-defined function, which must be declared as taking no parameters and returning type trigger. This is executed when a trigger fires.

- **arguments**

  Specifies an optional comma-separated list of parameters to be provided to the function when the trigger is executed. The parameters are literal string constants. Simple names and numeric constants can also be written here, but they will all be converted to strings. Check the description of the implementation language of the trigger function to find out how these parameters can be accessed within the function.

  NOTE

  The following details trigger types:

  - **INSTEAD OF** triggers must be marked as **FOR EACH ROW** and can be defined only on views.
  - **BEFORE** and **AFTER** triggers on a view must be marked as **FOR EACH STATEMENT**.
  - **TRUNCATE** triggers must be marked as **FOR EACH STATEMENT**.

Table 11-120 Types of triggers supported on tables and views

When	Event	Row-Level	Statement-Level
BEFORE	INSERT/UPDATE/DELETE	Tables	Tables and views
	TRUNCATE	Not supported.	Tables
AFTER	INSERT/UPDATE/DELETE	Tables	Tables and views
	TRUNCATE	Not supported.	Tables
INSTEAD OF	INSERT/UPDATE/DELETE	Views	Not supported.
	TRUNCATE	Not supported.	Not supported.

Table 11-121 Special variables in PL/pgSQL functions

Variable	Description
NEW	New tuple for **INSERT** and **UPDATE** operations. This variable is **NULL** for **DELETE** operations.
OLD	Old tuple for **UPDATE** and **DELETE** operations. This variable is **NULL** for **INSERT** operations.
TG_NAME	Trigger name.
TG_WHEN	Trigger timing (**BEFORE**, **AFTER**, or **INSTEAD OF**).
TG_LEVEL	Trigger frequency (**ROW** or **STATEMENT**).
TG_OP	Trigger event (**INSERT**, **UPDATE**, **DELETE**, or **TRUNCATE**).
TG_RELID	OID of the table where the trigger resides.
TG_RELNAME	Name of the table where the trigger resides. (This variable has been replaced by **TG_TABLE_NAME**.)
TG_TABLE_NAME	Name of the table where the trigger resides.
TG_TABLE_SCHEMA	Schema of the table where the trigger resides.
TG_NARGS	Number of parameters for the trigger function.
TG_ARGV[]	List of parameters for the trigger function.

Examples

```
-- Create a source table and a destination table.
openGauss=# CREATE TABLE test_trigger_src_tbl(id1 INT, id2 INT, id3 INT);
openGauss=# CREATE TABLE test_trigger_des_tbl(id1 INT, id2 INT, id3 INT);

-- Create a trigger function.
openGauss=# CREATE OR REPLACE FUNCTION tri_insert_func() RETURNS TRIGGER AS
           $$
           DECLARE
           BEGIN
                   INSERT INTO test_trigger_des_tbl VALUES(NEW.id1, NEW.id2, NEW.id3);
                   RETURN NEW;
           END
           $$ LANGUAGE PLPGSQL;

openGauss=# CREATE OR REPLACE FUNCTION tri_update_func() RETURNS TRIGGER AS
           $$
```

```
          DECLARE
          BEGIN
                 UPDATE test_trigger_des_tbl SET id3 = NEW.id3 WHERE id1=OLD.id1;
                 RETURN OLD;
          END
          $$ LANGUAGE PLPGSQL;

openGauss=# CREATE OR REPLACE FUNCTION TRI_DELETE_FUNC() RETURNS TRIGGER AS
          $$
          DECLARE
          BEGIN
                 DELETE FROM test_trigger_des_tbl WHERE id1=OLD.id1;
                 RETURN OLD;
          END
          $$ LANGUAGE PLPGSQL;

-- Create an INSERT trigger.
openGauss=# CREATE TRIGGER insert_trigger
          BEFORE INSERT ON test_trigger_src_tbl
          FOR EACH ROW
          EXECUTE PROCEDURE tri_insert_func();

-- Create an UPDATE trigger.
openGauss=# CREATE TRIGGER update_trigger
          AFTER UPDATE ON test_trigger_src_tbl
          FOR EACH ROW
          EXECUTE PROCEDURE tri_update_func();

-- Create a DELETE trigger.
openGauss=# CREATE TRIGGER delete_trigger
          BEFORE DELETE ON test_trigger_src_tbl
          FOR EACH ROW
          EXECUTE PROCEDURE tri_delete_func();

-- Execute the INSERT event and check the trigger results.
openGauss=# INSERT INTO test_trigger_src_tbl VALUES(100,200,300);
openGauss=# SELECT * FROM test_trigger_src_tbl;
openGauss=# SELECT * FROM test_trigger_des_tbl;  // Check whether the trigger operation takes effect.

-- Execute the UPDATE event and check the trigger results.
openGauss=# UPDATE test_trigger_src_tbl SET id3=400 WHERE id1=100;
openGauss=# SELECT * FROM test_trigger_src_tbl;
openGauss=# SELECT * FROM test_trigger_des_tbl;  // Check whether the trigger operation takes effect.

-- Execute the DELETE event and check the trigger results.
openGauss=# DELETE FROM test_trigger_src_tbl WHERE id1=100;
openGauss=# SELECT * FROM test_trigger_src_tbl;
openGauss=# SELECT * FROM test_trigger_des_tbl;  // Check whether the trigger operation takes effect.

-- Modify a trigger.
openGauss=# ALTER TRIGGER delete_trigger ON test_trigger_src_tbl RENAME TO delete_trigger_renamed;

-- Disable insert_trigger.
openGauss=# ALTER TABLE test_trigger_src_tbl DISABLE TRIGGER insert_trigger;

-- Disable all triggers on the current table.
openGauss=# ALTER TABLE test_trigger_src_tbl DISABLE TRIGGER ALL;

-- Delete triggers.
openGauss=# DROP TRIGGER insert_trigger ON test_trigger_src_tbl;
openGauss=# DROP TRIGGER update_trigger ON test_trigger_src_tbl;
openGauss=# DROP TRIGGER delete_trigger_renamed ON test_trigger_src_tbl;
```

Helpful Links

ALTER TRIGGER, **DROP TRIGGER**, and **ALTER TABLE**

11.14.94 CREATE TYPE

Function

CREATE TYPE registers a new data type for use in the current database. The user who defines a type becomes its owner. Types are designed only for row-store tables.

The following data types can be created: composite type, base type, shell type, enumerated type, and set type.

- Composite type

  A composite type is specified by a list of attribute names and data types. If the data type of an attribute is collatable, the attribute's collation rule can also be specified. This is essentially the same as the row type of a table, but using **CREATE TYPE** avoids the need to create an actual table when all that is wanted is to define a type. A stand-alone composite type is useful as the parameter or return type of a function.

  To create a composite type, you must have the **USAGE** permission on all of its attribute types.
- Base type

  You can create a base type (scalar type). Generally, these functions must be written in the underlying language.
- Shell type

  A shell type is simply a placeholder for a type to be defined later; it is created by issuing **CREATE TYPE** with no parameters except for the type name. Shell types are needed as forward references when base types are created.
- Enumerated type

  An enumerated type is a list of one or more quoted labels, each of which must be 1 to 63 bytes long.
- Set type

  It is similar to an array but has no length limit. It is mainly used in stored procedures.
- A user granted with the **CREATE ANY TYPE** permission can create types in the public and user schemas.

Precautions

If a schema name is given then the type is created in the specified schema. Otherwise, it is created in the current schema. The type name must be distinct from the name of any existing type or domain in the same schema. (Because tables have associated data types, the type name must also be distinct from the name of any existing table in the same schema.)

Syntax

```
CREATE TYPE name AS
    ( [ attribute_name data_type [ COLLATE collation ] [, ... ] ] )

CREATE TYPE name (
    INPUT = input_function,
    OUTPUT = output_function
    [ , RECEIVE = receive_function ]
    [ , SEND = send_function ]
    [ , TYPMOD_IN =
type_modifier_input_function ]
    [ , TYPMOD_OUT =
type_modifier_output_function ]
    [ , ANALYZE = analyze_function ]
    [ , INTERNALLENGTH = { internallength |
VARIABLE } ]
    [ , PASSEDBYVALUE ]
    [ , ALIGNMENT = alignment ]
  [ , STORAGE = storage ]
    [ , LIKE = like_type ]
    [ , CATEGORY = category ]
    [ , PREFERRED = preferred ]
    [ , DEFAULT = default ]
    [ , ELEMENT = element ]
    [ , DELIMITER = delimiter ]
    [ , COLLATABLE = collatable ]
)

CREATE TYPE name

CREATE TYPE name AS ENUM
    ( [ 'label' [, ... ] ] )

CREATE TYPE name AS TABLE OF data_type
```

Parameter Description

Composite type

- **name**

  Specifies the name (optionally schema-qualified) of the type to be created.

- **attribute_name**

  Specifies the name of an attribute (column) for the composite type.

- **data_type**

  Specifies the name of an existing data type to become a column of the composite type. You can use **%ROWTYPE** to indirectly reference the type of a table, or **%TYPE** to indirectly reference the type of a column in a table or composite type.

- **collation**

  Specifies the name of an existing collation rule to be associated with a column of the composite type. You can run the **select * from pg_collation** command to query collation rules from the **pg_collation** system catalog. The default collation rule is the row starting with **default** in the query result.

Base type

When creating a base type, you can place parameters in any order. The **input_function** and **output_function** parameters are mandatory, and other parameters are optional.

- **input_function**

  Specifies the name of a function that converts data from the type's external textual form to its internal form.

  The input function may be declared as taking one parameter of type cstring or taking three parameters of types cstring, oid, and integer.

- The first parameter is the input text as a C string,
- the second parameter is the type's own OID (except for array types, which instead receive their element type's OID),
- and the third is the typmod of the destination column, if known (**-1** will be passed if not).

The input function must return a value of the data type itself. Usually, an input function should be declared **STRICT**; if it is not, it will be called with a **NULL** first parameter when reading a **NULL** input value. The function must still return **NULL** in this case, unless it raises an error. (This case is mainly meant to support domain input functions, which might need to reject **NULL** inputs.)

NOTE

The input and output functions can be declared to have results or parameters of the new type, when they have to be created before the new type can be created. The type should first be defined as a shell type, which is a placeholder type that has no attributes except a name and an owner. This is done by issuing the **CREATE TYPE** *name* statement, with no additional parameters. Then the I/O functions can be defined referencing the shell type. Finally, **CREATE TYPE** with a full definition replaces the shell entry with a complete, valid type definition, after which the new type can be used normally.

- **output_function**

  Specifies the name of a function that converts data from the type's internal form to its external textual form.

  The output function must be declared as taking one parameter of the new data type. The output function must return type cstring. Output functions are not invoked for **NULL** values.

- **receive_function**

  (Optional) Specifies the name of a function that converts data from the type's external binary form to its internal form.

  If this function is not supplied, the type cannot participate in binary input. The binary representation should be chosen to be cheap to convert to internal form, while being reasonably portable. (For example, the standard integer data types use network byte order as the external binary representation, while the internal representation is in the machine's native byte order.) The receive function should perform adequate checking to ensure that the value is valid.

  The receive function may be declared as taking one parameter of type internal or taking three parameters of types internal, oid, integer.

  - The first parameter is a pointer to a StringInfo buffer holding the received byte string;
  - the latter two are the same as for the text input function.

  The receive function must return a value of the data type itself. Usually, a receive function should be declared **STRICT**; if it is not, it will be called with a **NULL** first parameter when reading a **NULL** input value. The function must still return **NULL** in this case, unless it raises an error. (This case is mainly meant to support domain receive functions, which might need to reject **NULL** inputs.)

- **send_function**

  (Optional) Specifies the name of a function that converts data from the type's internal form to its external binary form.

If this function is not supplied, the type cannot participate in binary output. The send function must be declared as taking one parameter of the new data type. The send function must return type bytea. Send functions are not invoked for **NULL** values.

- **type_modifier_input_function**

  (Optional) Specifies the name of a function that converts an array of modifier(s) for the type to an internal form.

- **type_modifier_output_function**

  (Optional) Specifies the name of a function that converts the internal form of the type's modifier(s) to its external textual form.

  NOTE

  type_modifier_input_function and **type_modifier_output_function** are needed if the type supports modifiers, that is optional constraints attached to a type declaration, such as char(5) or numeric(30,2). GaussDB allows user-defined types to take one or more simple constants or identifiers as modifiers. However, this information must be capable of being packed into a single non-negative integer value for storage in the system catalogs. The **type_modifier_input_function** is passed the declared modifier(s) in the form of a cstring array. It must check the values for validity (throwing an error if they are wrong), and if they are correct, return a single non-negative integer value that will be stored as the column "typmod". Type modifiers will be rejected if the type does not have a **type_modifier_input_function**. The **type_modifier_output_function** converts the internal integer typmod value back to the correct form for user display. It must return a cstring value that is the exact string to append to the type name; for example numeric's function might return (30,2). It is allowed to omit the **type_modifier_output_function**, in which case the default display format is just the stored typmod integer value enclosed in parentheses.

- **analyze_function**

  (Optional) Specifies the name of a function that performs statistical analysis for the data type.

  By default, **ANALYZE** will attempt to gather statistics using the type's "equals" and "less-than" operators, if there is a default B-tree operator class for the type. For non-scalar types, this behavior is likely to be unsuitable, so it can be overridden by specifying a custom analysis function. The analysis function must be declared to take one parameter of type internal and return a boolean result.

- **internallength**

  (Optional) Specifies the length in bytes of the new type's internal representation. The default assumption is that it is variable-length.

  While the details of the new type's internal representation are only known to the I/O functions and other functions you create to work with the type, there are several attributes of the internal representation that must be declared to GaussDB. Foremost of these is **internallength**. Base data types can be fixed-length, in which case **internallength** is a positive integer, or variable length, indicated by setting **internallength** to **VARIABLE**. (Internally, this is represented by setting **typlen** to **-1**.) The internal representation of all variable-length types must start with a 4-byte integer giving the total length of this value of the type.

- **PASSEDBYVALUE**

  (Optional) Indicates that values of this data type are passed by value, rather than by reference. You cannot pass by value types whose internal representation is larger than the size of the Datum type (4 bytes on most machines, 8 bytes on a few).

- **alignment**

  (Optional) Specifies the storage alignment requirement of the data type. If specified, it must be **char**, **int2**, **int4**, or **double**; the default is **int4**.

  The allowed values equate to alignment on 1, 2, 4, or 8 byte boundaries. Note that variable-length types must have an alignment of at least 4, since they necessarily contain an int4 as their first component.

- **storage**

  (Optional) Specifies the storage strategy for the data type.

  If specified, it must be **plain**, **external**, **extended**, or **main**; the default is **plain**.

  - **plain** specifies that data of the type will always be stored in-line and not compressed. (Only **plain** is allowed for fixed-length types.)
  - **extended** specifies that the system will first try to compress a long data value, and will move the value out of the main table row if it is still too long.
  - **external** allows the value to be moved out of the main table, but the system will not try to compress it.
  - **main** allows compression, but discourages moving the value out of the main table. (Data items with this storage strategy might still be moved out of the main table if there is no other way to make a row fit, but they will be kept in the main table preferentially over **extended** and **external** items.)

    All **storage** values other than **plain** imply that the functions of the data type can handle values that have been toasted. The specific other value given merely determines the default **TOAST** storage strategy for columns of a toastable data type; users can pick other strategies for individual columns using **ALTER TABLE SET STORAGE**.

- **like_type**

  (Optional) Specifies the name of an existing data type that the new type will have the same representation as. The values of **internallength**, **passedbyvalue**, **alignment**, and **storage** are copied from that type, unless overridden by explicit specification elsewhere in this **CREATE TYPE** statement.

  Specifying representation in this way is especially useful when the low-level implementation of a new type references an existing type.

- **category**

  (Optional) Specifies the category code (a single ASCII character) for this type. The default is **U** for a user-defined type. You may also choose other ASCII characters to create custom categories.

- **preferred**

  (Optional) Specifies whether a type is preferred within its type category. If it is, the value will be **TRUE**, else **FALSE**. The default is **FALSE**. Be very careful about creating a preferred type within an existing type category, as this could cause surprising changes in behavior.

NOTE

The **category** and **preferred** parameters can be used to help control which implicit cast will be applied in ambiguous situations. Each data type belongs to a category named by a single ASCII character, and each type is either preferred or not within its category. The parser will prefer casting to preferred types (but only from other types within the same category) when this rule is helpful in resolving overloaded functions or operators. For types that have no implicit casts to or from any other types, it is sufficient to leave these settings at the defaults. However, for a group of related types that have implicit casts, it is often helpful to mark them all as belonging to a category and select one or two of the most general types as being preferred within the category. The **category** parameter is especially useful when adding a user-defined type to an existing built-in category, such as the numeric or string types. However, it is also possible to create entirely-user-defined type categories. Select any ASCII character other than an uppercase letter to name such a category.

- **default**

  (Optional) Specifies the default value for the data type. If this is omitted, the default is null.

  A default value can be specified, in case a user wants columns of the data type to default to something other than the null value. Specify the default with the **DEFAULT** keyword. (Such a default can be overridden by an explicit **DEFAULT** clause attached to a particular column.)

- **element**

  (Optional) Specifies the type of array elements when an array type is created. For example, to define an array of 4-byte integers (int4), specify **ELEMENT = int4**.

- **delimiter**

  (Optional) Specifies the delimiter character to be used between values in arrays made of this type.

  delimiter can be set to a specific character. The default delimiter is the comma (,). Note that the delimiter is associated with the array element type, not the array type itself.

- **collatable**

  (Optional) Specifies whether this type's operations can use collation information. If they can, the value will be **TRUE**, else **FALSE** (default).

  If **collatable** is **TRUE**, column definitions and expressions of the type may carry collation information through use of the **COLLATE** clause. It is up to the implementations of the functions operating on the type to actually make use of the collation information; this does not happen automatically merely by marking the type collatable.

- **label**

  (Optional) Represents the textual label associated with one value of an enumerated type. It is a string of 1 to 63 characters.

NOTE

Whenever a user-defined type is created, GaussDB automatically creates an associated array type whose name consists of the element type's name prepended with an underscore (_).

Examples

```
-- Create a composite type, create a table, insert data, and make a query.
openGauss=# CREATE TYPE compfoo AS (f1 int, f2 text);
```

```
openGauss=# CREATE TABLE t1_compfoo(a int, b compfoo);
openGauss=# CREATE TABLE t2_compfoo(a int, b compfoo);
openGauss=# INSERT INTO t1_compfoo values(1,(1,'demo'));
openGauss=# INSERT INTO t2_compfoo select * from t1_compfoo;
openGauss=# SELECT (b).f1 FROM t1_compfoo;
openGauss=# SELECT * FROM t1_compfoo t1 join t2_compfoo t2 on (t1.b).f1=(t1.b).f1;

-- Rename the data type.
openGauss=# ALTER TYPE compfoo RENAME TO compfoo1;

-- Change the owner of the user-defined type compfoo1 to usr1.
openGauss=# CREATE USER usr1 PASSWORD 'xxxxxxxxx';
openGauss=# ALTER TYPE compfoo1 OWNER TO usr1;

-- Change the schema of the user-defined type compfoo1 to usr1.
openGauss=# ALTER TYPE compfoo1 SET SCHEMA usr1;

-- Add a new attribute to the data type.
openGauss=# ALTER TYPE usr1.compfoo1 ADD ATTRIBUTE f3 int;

-- Delete the compfoo1 type.
openGauss=# DROP TYPE usr1.compfoo1 cascade;

-- Delete related tables and users.
openGauss=# DROP TABLE t1_compfoo;
openGauss=# DROP TABLE t2_compfoo;
openGauss=# DROP SCHEMA usr1;
openGauss=# DROP USER usr1;

-- Create an enumerated type.
openGauss=# CREATE TYPE bugstatus AS ENUM ('create', 'modify', 'closed');

-- Add a label.
openGauss=# ALTER TYPE bugstatus ADD VALUE IF NOT EXISTS 'regress' BEFORE 'closed';

-- Rename a label.
openGauss=# ALTER TYPE bugstatus RENAME VALUE 'create' TO 'new';

-- Create a set type.
openGauss=# CREATE TYPE compfoo_table AS TABLE OF compfoo;
```

Helpful Links

ALTER TYPE and **DROP TYPE**

11.14.95 CREATE USER

Function

CREATE USER creates a user.

Precautions

- A user created using the CREATE USER statement has the LOGIN permission by default.
- When you run the **CREATE USER** command to create a user, the system creates a schema with the same name as the user in the database where the command is executed.
- The owner of an object created by a system administrator in a schema with the same name as a common user is the common user, not the system administrator.

Syntax

```
CREATE USER user_name [ [ WITH ] option [ ... ] ] [ ENCRYPTED | UNENCRYPTED ] { PASSWORD |
IDENTIFIED BY } { 'password' [EXPIRED] | DISABLE };
```

The **option** clause is used to configure information, including permissions and properties.

```
{SYSADMIN | NOSYSADMIN}
    | {MONADMIN | NOMONADMIN}
    | {OPRADMIN | NOOPRADMIN}
    | {POLADMIN | NOPOLADMIN}
    | {AUDITADMIN | NOAUDITADMIN}
    | {CREATEDB | NOCREATEDB}
    | {USEFT | NOUSEFT}
    | {CREATEROLE | NOCREATEROLE}
    | {INHERIT | NOINHERIT}
    | {LOGIN | NOLOGIN}
    | {REPLICATION | NOREPLICATION}
    | {INDEPENDENT | NOINDEPENDENT}
    | {VCADMIN | NOVCADMIN}
    | {PERSISTENCE | NOPERSISTENCE}
    | CONNECTION LIMIT connlimit
    | VALID BEGIN 'timestamp'
    | VALID UNTIL 'timestamp'
    | RESOURCE POOL 'respool'
    | USER GROUP 'groupuser'
    | PERM SPACE 'spacelimit'
    | TEMP SPACE 'tmpspacelimit'
    | SPILL SPACE 'spillspacelimit'
    | NODE GROUP logic_cluster_name
    | IN ROLE role_name [, ...]
    | IN GROUP role_name [, ...]
    | ROLE role_name [, ...]
    | ADMIN role_name [, ...]
    | USER role_name [, ...]
    | SYSID uid
    | DEFAULT TABLESPACE tablespace_name
    | PROFILE DEFAULT
    | PROFILE profile_name
    | PGUSER
```

Parameter Description

- **user_name**

  Username.

  Value range: a string. It must comply with the identifier naming convention. A value can contain a maximum of 63 characters.

- **password**

  Specifies the login password.

  The new password must:

  - Contain at least eight characters. This is the default length.
  - Differ from the username or the username spelled backward.
  - Contain at least three of the following character types: uppercase characters, lowercase characters, digits, and special characters (limited to ~!@#$ %^&*()-_=+\|[{}];:,<.>/?).
  - The password can also be a ciphertext character string that meets the format requirements. This mode is mainly used to import user data. You are not advised to use it directly. If a ciphertext password is used, the user must know the plaintext corresponding to the ciphertext password and

ensure that the plaintext password meets the complexity requirements. The database does not verify the complexity of the ciphertext password. Instead, the security of the ciphertext password is ensured by the user.

– Be enclosed by single quotation marks when a user is created.

Value range: a string

For other parameters, see **CREATE ROLE**.

Examples

```
-- Create user jim whose login password is xxxxxxxxx:
openGauss=# CREATE USER jim PASSWORD 'xxxxxxxxx';

-- Alternatively, you can run the following statement:
openGauss=# CREATE USER kim IDENTIFIED BY 'xxxxxxxxx';

-- To create a user with the CREATEDB permission, add the CREATEDB keyword.
openGauss=# CREATE USER dim CREATEDB PASSWORD 'xxxxxxxxx';

-- Change user jim's login password from xxxxxxxxx to Abcd@123:
openGauss=# ALTER USER jim IDENTIFIED BY 'Abcd@123' REPLACE 'xxxxxxxxx';

-- Add the CREATEROLE permission to jim.
openGauss=# ALTER USER jim CREATEROLE;

-- Set enable_seqscan to on. (The setting will take effect in the next session.)
openGauss=# ALTER USER jim SET enable_seqscan TO on;

-- Reset the enable_seqscan parameter for jim.
openGauss=# ALTER USER jim RESET enable_seqscan;

-- Lock jim.
openGauss=# ALTER USER jim ACCOUNT LOCK;

-- Delete users.
openGauss=# DROP USER kim CASCADE;
openGauss=# DROP USER jim CASCADE;
openGauss=# DROP USER dim CASCADE;
```

Helpful Links

ALTER USER, **CREATE ROLE**, and **DROP USER**

11.14.96 CREATE USER MAPPING

Function

CREATE USER MAPPING defines a new mapping from a user to a foreign server.

Precautions

If the **password** option is displayed, ensure that the **usermapping.key.cipher** and **usermapping.key.rand** files exist in the *$GAUSSHOME***/bin** directory of each node in GaussDB. If the two files do not exist, use the gs_guc tool to generate them and use the gs_ssh tool to release them to the *$GAUSSHOME***/bin** directory on each node in GaussDB.

When multi-layer quotation marks are used for sensitive columns (such as **password**) in OPTIONS, the semantics is different from that in the scenario where quotation marks are not used. Sensitive columns are not identified for anonymization.

Syntax

```
CREATE USER MAPPING FOR { user_name | USER | CURRENT_USER | PUBLIC }
    SERVER server_name
    [ OPTIONS ( option 'value' [ , ... ] ) ]
```

Parameter Description

- **user_name**

  Specifies the name of an existing user to map to a foreign server.

  CURRENT_USER and **USER** match the name of the current user. When **PUBLIC** is specified, a public mapping is created and used when no mapping for a particular user is available.

- **server_name**

  Specifies the name of the existing server for which a user mapping will be created.

- **OPTIONS ({ option_name ' value ' } [, ...])**

  Specifies options for user mapping. These options typically define the actual user name and password for this mapping. The option name must be unique. The allowed option names and values are related to the foreign data wrapper of the server.

  NOTE

  - User passwords are encrypted and stored in the system catalog PG_USER_MAPPING. During the encryption, **usermapping.key.cipher** and **usermapping.key.rand** are used as the encryption password file and encryption factor. Before using the tool for the first time, run the following command to create the two files, save the files to the *$GAUSSHOME*/**bin** directory on each node, and ensure that you have the read permission on the files. **gs_ssh** helps you quickly place files in the specified directory of each node.
    ```
    gs_ssh -c "gs_guc generate -o usermapping -S default -D $GAUSSHOME/bin"
    ```
  - If the **-S** parameter is set to default, a password is randomly generated. You can also specify a password for the **-S** parameter to ensure the security and uniqueness of the generated password file. You do not need to save or memorize the password. For details about other parameters, see the description of the **gs_guc** tool in the "Tool Reference".

Helpful Links

ALTER USER MAPPING and DROP USER MAPPING.

11.14.97 CREATE VIEW

Function

CREATE VIEW creates a view. A view is a virtual table, not a base table. Only view definition is stored in the database and view data is not. The data is stored in a base table. If data in the base table changes, the data in the view changes accordingly. In this sense, a view is like a window through which users can know their interested data and data changes in the database.

Precautions

A user granted with the CREATE ANY TABLE permission can create views in the public and user schemas.

Syntax

```
CREATE [ OR REPLACE ] [ TEMP | TEMPORARY ] VIEW view_name [ ( column_name [, ...] ) ]
    [ WITH ( {view_option_name [= view_option_value]} [, ... ] ) ]
    AS query;
```

NOTE

You can use **WITH(security_barrier)** to create a relatively secure view. This prevents attackers from printing hidden base table data by using the **RAISE** statement of low-cost functions.

After a view is created, you are not allowed to use REPLACE to modify column names in the view or delete the columns.

Parameter Description

- **OR REPLACE**

  Redefines the view if it already exists.

- **TEMP | TEMPORARY**

  Creates a temporary view.

- **view_name**

  Specifies the name (optionally schema-qualified) of the view to be created.

  Value range: a string. It must comply with the identifier naming convention.

- **column_name**

  Specifies an optional list of names to be used for columns of the view. If not given, the column names are deduced from the query.

  Value range: a string. It must comply with the identifier naming convention.

- **view_option_name [= view_option_value]**

  Specifies an optional parameter for a view.

  Currently, **view_option_name** supports only the **security_barrier** parameter. This parameter is used when the view attempts to provide row-level security.

  Value range: Boolean type, **TRUE**, and **FALSE**.

- **query**

  Specifies a **SELECT** or **VALUES** statement that will provide the columns and rows of the view.

NOTICE

If **query** contains a clause specifying the partition of a partitioned table, the OID of the specified partition is hardcoded to the system catalog when the view is created. If a DDL operation that will change the OID of the specified partition is used, for example, DROP, SPLIT, or MERGE, the view is unavailable. In this case, you need to create a view.

Examples

```
-- Create a view consisting of columns whose spcname is pg_default.
openGauss=# CREATE VIEW myView AS
    SELECT * FROM pg_tablespace WHERE spcname = 'pg_default';

-- Query a view.
openGauss=# SELECT * FROM myView ;

-- Delete the myView view.
openGauss=# DROP VIEW myView;
```

Helpful Links

ALTER VIEW and **DROP VIEW**

11.14.98 CREATE WEAK PASSWORD DICTIONARY

Function

CREATE WEAK PASSWORD DICTIONARY inserts one or more weak passwords into the **gs_global_config** table.

Precautions

- Only the initial user, system administrator, and security administrator have the permission to execute this syntax.
- Passwords in the weak password dictionary are stored in the **gs_global_config** system catalog.
- The weak password dictionary is empty by default. You can use this syntax to add one or more weak passwords.
- When a user attempts to execute this syntax to insert a weak password that already exists in the **gs_global_config** table, only one weak password is retained in the table.

Syntax

```
CREATE WEAK PASSWORD DICTIONARY
       [WITH VALUES] ( {'weak_password'} [, ...] );
```

Parameter Description

weak_password

Specifies a weak password.

Value range: a character string

Examples

```
-- Insert a single weak password into the gs_global_config system catalog.
openGauss=# CREATE WEAK PASSWORD DICTIONARY WITH VALUES ('password1');

-- Insert multiple weak passwords into the gs_global_config system catalog.
openGauss=# CREATE WEAK PASSWORD DICTIONARY WITH VALUES ('password2'),('password3');

-- Clear all weak passwords in the gs_global_config system catalog.
openGauss=# DROP WEAK PASSWORD DICTIONARY;
```

```
-- View existing weak passwords.
openGauss=# SELECT * FROM gs_global_config WHERE NAME LIKE 'weak_password';
```

Helpful Links

DROP WEAK PASSWORD DICTIONARY

11.14.99 CURSOR

Function

CURSOR defines a cursor to retrieve a small number of rows at a time out of a larger query.

To process SQL statements, the stored procedure process assigns a memory segment to store context association. Cursors are handles or pointers pointing to context regions. With cursors, stored procedures can control alterations in context regions.

Precautions

- **CURSOR** is used only in transaction blocks.
- Generally, **CURSOR** and **SELECT** both have text returns. Since data is stored in binary format in the system, the system needs to convert the data from the binary format to the text format. If data is returned in text format, client applications need to convert the data back to the binary format for processing. **FETCH** implements conversion between binary data and text data.
- Binary cursors should be used carefully. Text usually occupies larger space than binary data. A binary cursor returns internal binary data, which is easier to operate. A text cursor returns text, which is easier to retrieve and therefore reduces workload on the client. As an example, if a query returns a value of one from an integer column, you would get a string of 1 with a default cursor, whereas with a binary cursor you would get a 4-byte field containing the internal representation of the value (in big-endian byte order).

Syntax

```
CURSOR cursor_name
    [ BINARY ]  [ NO SCROLL ]  [ { WITH | WITHOUT } HOLD ]
    FOR query ;
```

Parameter Description

- **cursor_name**

  Specifies the name of the cursor to be created.

  Value range: a string. It must comply with the naming convention.
- **BINARY**

  Causes the cursor to return data in binary rather than in text format.
- **NO SCROLL**

  Specifies how the cursor retrieves rows.

  – **NO SCROLL**: specifies that the cursor cannot be used to retrieve rows in a nonsequential fashion.

- Unspecified: Based on the query's execution plan, the system automatically determines whether the cursor can be used to retrieve rows in a nonsequential fashion.

- **WITH HOLD | WITHOUT HOLD**

  Specifies whether the cursor can continue to be used after the transaction that created it successfully commits.

  - **WITH HOLD**: The cursor can continue to be used after the transaction that created it successfully commits.
  - **WITHOUT HOLD**: The cursor cannot be used outside of the transaction that created it.
  - If neither **WITH HOLD** nor **WITHOUT HOLD** is specified, the default is **WITHOUT HOLD**.
  - Cross-node transactions (for example, DDL-contained transactions created in the database with multiple DBnode) do not support **WITH HOLD**.

- **query**

  Uses a **SELECT** or **VALUES** clause to specify the rows to be returned by the cursor.

  Value range: **SELECT** or **VALUES** clause

Examples

See **Examples** in **FETCH**.

Helpful Links

FETCH

11.14.100 DEALLOCATE

Function

DEALLOCATE deallocates a previously prepared statement. If you do not explicitly deallocate a prepared statement, it is deallocated when the session ends.

The **PREPARE** keyword is always ignored.

Precautions

None

Syntax

```
DEALLOCATE [ PREPARE ] { name | ALL };
```

Parameter Description

- **name**

  Specifies the name of the prepared statement to be deallocated.

- **ALL**

Deallocates all prepared statements.

Examples

None

11.14.101 DECLARE

Function

DECLARE defines a cursor to retrieve a small number of rows at a time out of a larger query and can be the start of an anonymous block.

This section describes usage of cursors. The usage of anonymous blocks is available in **BEGIN**.

To process SQL statements, the stored procedure process assigns a memory segment to store context association. Cursors are handles or pointers pointing to context regions. With cursors, stored procedures can control alterations in context regions.

Generally, **CURSOR** and **SELECT** both have text returns. Since data is stored in binary format in the system, the system needs to convert the data from the binary format to the text format. If data is returned in text format, client applications need to convert the data back to the binary format for processing. **FETCH** implements conversion between binary data and text data.

Precautions

- **CURSOR** is used only in transaction blocks.
- Binary cursors should be used carefully. Text usually occupies larger space than binary data. A binary cursor returns internal binary data, which is easier to operate. A text cursor returns text, which is easier to retrieve and therefore reduces workload on the client. As an example, if a query returns a value of one from an integer column, you would get a string of 1 with a default cursor, whereas with a binary cursor you would get a 4-byte field containing the internal representation of the value (in big-endian byte order).

Syntax

- Define a cursor.
```
DECLARE cursor_name [ BINARY ] [ NO SCROLL ]
    CURSOR [ { WITH | WITHOUT } HOLD ] FOR query ;
```
- Enable an anonymous block.
```
[DECLARE [declare_statements]]
BEGIN
execution_statements
END;
/
```

Parameter Description

- **cursor_name**

  Specifies the name of the cursor to be created.

  Value range: a string. It must comply with the naming convention.

- **BINARY**

  Causes the cursor to return data in binary rather than in text format.
- **NO SCROLL**

  Specifies how the cursor retrieves rows.
  - **NO SCROLL**: specifies that the cursor cannot be used to retrieve rows in a nonsequential fashion.
  - Unspecified: Based on the query's execution plan, the system automatically determines whether the cursor can be used to retrieve rows in a nonsequential fashion.
- **WITH HOLD**

  WITHOUT HOLD

  Specifies whether the cursor can continue to be used after the transaction that created it successfully commits.
  - **WITH HOLD**: The cursor can continue to be used after the transaction that created it successfully commits.
  - **WITHOUT HOLD**: The cursor cannot be used outside of the transaction that created it.
  - If neither **WITH HOLD** nor **WITHOUT HOLD** is specified, the default is **WITHOUT HOLD**.
- **query**

  Uses a **SELECT** or **VALUES** clause to specify the rows to be returned by the cursor.

  Value range: **SELECT** or **VALUES** clause
- **declare_statements**

  Declares a variable, including its name and type, for example, **sales_cnt int**.
- **execution_statements**

  Specifies the statement to be executed in an anonymous block.

  Value range: an existing function name

Examples

For details about how to define a cursor, see **Examples** in **FETCH**.

Helpful Links

BEGIN and **FETCH**

11.14.102 DELETE

Function

DELETE deletes rows that satisfy the **WHERE** clause from the specified table. If the **WHERE** clause is absent, the effect is to delete all rows in the table. The result is a valid, but an empty table.

Precautions

- The owner of a table, users granted with the **DELETE** permission on the table, or users granted with the **DELETE ANY TABLE** permission can delete data from the table. The system administrator has the permission to delete data from the table by default, as well as the **SELECT** permission on any table in the **USING** clause or whose values are read in **condition**.
- For column-store tables, the **RETURNING** clause is currently not supported.

Syntax

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
DELETE [/*+ plan_hint */] [FROM] [ ONLY ] table_name [partition_clause] [ * ] [ [ AS ] alias ]
    [ USING using_list ]
    [ WHERE condition | WHERE CURRENT OF cursor_name ]
    [ LIMIT { count } ]
    [ RETURNING { * | { output_expr [ [ AS ] output_name ] } [, ...] } ];
```

Parameter Description

- **WITH [RECURSIVE] with_query [, ...]**

  Specifies one or more subqueries that can be referenced by name in the main query, which is equivalent to a temporary table.

  If **RECURSIVE** is specified, it allows a **SELECT** subquery to reference itself by name.

  Format of **with_query**:

  ```
  with_query_name [ ( column_name [, ...] ) ] AS [ [ NOT ] MATERIALIZED ]
  ( {select | values | insert | update | delete} )
  ```

  -- **with_query_name** specifies the name of the result set generated by a subquery. Such names can be used to access the subquery

  result set.

  – **column_name** specifies the column name displayed in the subquery result set.

  – Each subquery can be a **SELECT**, **VALUES**, **INSERT**, **UPDATE** or **DELETE** statement.

  – You can use **MATERIALIZED** or **NOT MATERIALIZED** to modify the CTE.

  – If **MATERIALIZED** is specified, the WITH query will be materialized, and a copy of the subquery result set is generated. The copy is directly queried at the reference point. Therefore, the WITH subquery cannot be jointly optimized with the SELECT statement trunk (for example, predicate pushdown and equivalence class transfer). In this scenario, you can use **NOT MATERIALIZED** for modification. If the WITH query can be executed as a subquery inline, the preceding optimization can be performed.

  – If the user does not explicitly declare the materialized attribute, comply with the following rules: If the CTE is referenced only once in the trunk statement to which it belongs and semantically supports inline execution, it will be rewritten as subquery inline execution. Otherwise, the materialized execution will be performed in CTE Scan mode.

- **plan_hint** clause

  Follows the **DELETE** keyword in the **/*+ */** format. It is used to optimize the plan of a **DELETE** statement block. For details, see **Hint-based Tuning**. In each statement, only the first **/*+** *plan_hint* ***/** comment block takes effect as a hint. Multiple hints can be written.

- **ONLY**

  If **ONLY** is specified before the table name, matching rows are deleted from the named table only. If **ONLY** is not specified, matching rows are also deleted from any tables inheriting from the named table.

- **table_name**

  Specifies the name (optionally schema-qualified) of the target table.

  Value range: an existing table name

- **partition_clause**

  Deletes a specified partition.

  PARTITION { (partition_name) | FOR (partition_value [, ...]) } |

  SUBPARTITION { (subpartition_name) | FOR (subpartition_value [, ...]) }

  For details about the keywords, see **SELECT**.

  For details, see **CREATE TABLE SUBPARTITION**.

- **alias**

  Specifies a substitute name for the target table.

  Value range: a string. It must comply with the identifier naming convention.

- **using_list**

  Specifies the **USING** clause.

- **condition**

  Specifies an expression that returns a Boolean value. Only rows for which this expression returns **true** will be deleted. You are not advised to use numeric types such as int for **condition**, because such types can be implicitly converted to bool values (non-zero values are implicitly converted to **true** and **0** is implicitly converted to **false**), which may cause unexpected results.

- **WHERE CURRENT OF cursor_name**

  This parameter is reserved.

- **LIMIT**

  For details about the keywords, see **SELECT**.

- **output_expr**

  Specifies an expression to be computed and returned by the **DELETE** statement after each row is deleted. The expression can use any column names of the table. Write ***** to return all columns.

- **output_name**

  Specifies a name to use for a returned column.

  Value range: a string. It must comply with the identifier naming convention.

Examples

```
-- Create the tpcds.customer_address_bak table.
openGauss=# CREATE TABLE tpcds.customer_address_bak AS TABLE tpcds.customer_address;

-- Delete employees whose ca_address_sk is smaller than 14888 from the tpcds.customer_address_bak
table.
openGauss=# DELETE FROM tpcds.customer_address_bak WHERE ca_address_sk < 14888;
```

```
-- Delete all data from the tpcds.customer_address_bak table.
openGauss=# DELETE FROM tpcds.customer_address_bak;

Delete the tpcds.customer_address_bak table.
openGauss=# DROP TABLE tpcds.customer_address_bak;
```

Suggestions

- delete

  To delete all records in a table, use the **truncate** syntax.

11.14.103 DO

Function

DO executes an anonymous code block.

The code block is treated as though it were the body of a function with no parameters, returning **void**. It is parsed and executed a single time.

Precautions

- The procedural language to be used must already have been installed into the current database by means of **CREATE LANGUAGE**. **plpgsql** is installed by default, but other languages are not.
- The user must have the **USAGE** permission on the procedural language, or must be a system administrator if the language is untrusted.

Syntax

```
DO [ LANGUAGE lang_name ] code;
```

Parameter Description

- **lang_name**

  Specifies the name of the procedural language the code is written in. If omitted, the default is **plpgsql**.
- **code**

  Specifies the procedural language code to be executed. This must be specified as a string literal.

Examples

```
-- Create the webuser user.
openGauss=# CREATE USER webuser PASSWORD 'xxxxxxxxx';

-- Grant all permissions on all views in the tpcds schema to the webuser user.
openGauss=# DO $$DECLARE r record;
BEGIN
    FOR r IN SELECT c.relname table_name,n.nspname table_schema FROM pg_class c,pg_namespace n
            WHERE c.relnamespace = n.oid AND n.nspname = 'tpcds' AND relkind IN ('r','v')
    LOOP
        EXECUTE 'GRANT ALL ON ' || quote_ident(r.table_schema) || '.' || quote_ident(r.table_name) || ' TO 
webuser';
    END LOOP;
END$$;
```

```
-- Delete the webuser user.
openGauss=# DROP USER webuser CASCADE;
```

11.14.104 DROP AGGREGATE

Function

DROP AGGREGATE deletes an aggregate function.

Precautions

DROP AGGREGATE deletes an existing aggregate function. Only the owner of the aggregate function can run this command.

Syntax

```
DROP AGGREGATE [ IF EXISTS ] name ( argtype [ , ... ] ) [ CASCADE | RESTRICT ]
```

Parameter Description

- **IF EXISTS**

  Do not throw an error if the specified aggregation does not exist. A notice is issued in this case.
- **name**

  Existing aggregate function name (optionally schema-qualified).
- **argtype**

  Input data type of the aggregate function. To reference a zero-parameter aggregate function, use * to replace the input data type list.
- **CASCADE**

  Cascade deletes objects that depend on the aggregate function.
- **RESTRICT**

  Refuses to delete the aggregate function if any objects depend on it. This is a default processing.

Examples

Delete the aggregate function **myavg** of the integer type:

```
DROP AGGREGATE myavg(integer);
```

Compatibility

The SQL standard does not provide the **DROP AGGREGATE** statement.

11.14.105 DROP AUDIT POLICY

Function

DROP AUDIT POLICY deletes an audit policy.

Precautions

Only user **poladmin**, user **sysadmin**, or the initial user can perform this operation.

Syntax

```
DROP AUDIT POLICY [IF EXISTS] policy_name;
```

Parameter Description

policy_name

Specifies the audit policy name, which must be unique.

Value range: a string. It must comply with the naming convention.

Examples

See **Examples** in **CREATE AUDIT POLICY**.

Helpful Links

ALTER AUDIT POLICY and **CREATE AUDIT POLICY**

11.14.106 DROP CAST

Function

DROP CAST deletes a type conversion.

Precautions

DROP CAST deletes a previously defined type conversion.

To delete a type conversion, you must have a source or destination data type. This is the same permission as creating a type conversion.

Syntax

```
DROP CAST [ IF EXISTS ] (source_type AS target_type) [ CASCADE | RESTRICT ]
```

Parameter Description

- **IF EXISTS**
  Do not throw an error if the specified conversion does not exist. A notice is issued in this case.
- **source_type**
  Source data type in the type conversion.
- **target_type**
  Type of the target data in the type conversion.
- **CASCADE**
  RESTRICT
  These keys have no effect because there is no dependency on type conversion.

Example

Delete the conversion from text to int.

```
DROP CAST (text AS int);
```

Compatibility

DROP CAST complies with the SQL standard.

11.14.107 DROP CLIENT MASTER KEY

Function

DROP CLIENT MASTER KEY deletes a client master key (CMK).

Precautions

- Only the CMK owner or a user who has been granted the DROP permission can run this command. By default, the system administrator has this permission.
- This command can only delete the metadata of key objects recorded in the system catalog of the database, but cannot delete the key entities managed by the client key tool or online key service.

Syntax

```
DROP CLIENT MASTER KEY [ IF EXISTS ] client_master_key_name [CASCADE];
```

Parameter Description

- **IF EXISTS**

  If a specified CMK does not exist, a notice rather than an error is issued.
- **client_master_key_name**

  Name of a CMK to be deleted.

  Value range: a string. It is the name of an existing CMK object.
- **CASCADE**
  - **CASCADE**: automatically deletes objects that depend on the CMK.

    NOTICE

Examples

```
-- Delete a CMK object.
openGauss=> DROP CLIENT MASTER KEY ImgCMK CASCADE;
NOTICE:  drop cascades to column setting: imgcek
DROP CLIENT MASTER KEY
```

11.14.108 DROP COLUMN ENCRYPTION KEY

Function

CREATE COLUMN ENCRYPTION KEY deletes a column encryption key (CEK).

Precautions

Only the CEK owner or a user who has been granted the DROP permission can run this command. By default, the system administrator has this permission.

Syntax

```
DROP COLUMN ENCRYPTION KEY [ IF EXISTS ] column_encryption_key_name [CASCADE];
```

Parameter Description

- **IF EXISTS**

  If a specified CEK does not exist, a notice rather than an error is issued.

- **column_encryption_key_name**

  Name of a CEK to be deleted.

  Value range: a string. It is the name of an existing CEK.

Examples

```
-- Delete a CEK.
openGauss=# DROP COLUMN ENCRYPTION KEY ImgCEK CASCADE;
ERROR:  cannot drop column setting: imgcek cascadely because encrypted column depend on it.
HINT:  we have to drop encrypted column: name, ... before drop column setting: imgcek cascadely.
```

11.14.109 DROP DATABASE

Function

DROP DATABASE deletes a database.

Precautions

- Only the database owner or a user granted with the DROP permission can run the **DROP DATABASE** command. The system administrator has this permission by default.
- The preinstalled POSTGRES, TEMPLATE0, and TEMPLATE1 databases are protected and therefore cannot be deleted. To check databases in the current service, run the gsql statement **\l**.
- If any users are connected to the database, the database cannot be deleted.
- **DROP DATABASE** cannot be executed within a transaction block.
- If **DROP DATABASE** fails and is rolled back, run **DROP DATABASE IF EXISTS** again.

NOTICE

DROP DATABASE cannot be undone.

Syntax

```
DROP DATABASE [ IF EXISTS ] database_name ;
```

Parameter Description

- **IF EXISTS**

  Reports a notice instead of an error if the specified database does not exist.
- **database_name**

  Specifies the name of the database to be deleted.

  Value range: an existing database name

Examples

See **Examples** in CREATE DATABASE.

Helpful Links

CREATE DATABASE

Suggestions

- drop database

  Do not delete databases during transactions.

11.14.110 DROP DATA SOURCE

Function

DROP DATA SOURCE deletes a data source.

Precautions

Only an owner, system administrator, or initial user can delete a data source.

Syntax

```
DROP DATA SOURCE [IF EXISTS] src_name [CASCADE | RESTRICT];
```

Parameter Description

- **src_name**

  Specifies the name of the data source to be deleted.

  Value range: a string. It must comply with the naming convention.
- **IF EXISTS**

  Reports a notice instead of an error if the specified data source does not exist.

- **CASCADE | RESTRICT**
  - **CASCADE**: automatically deletes the objects that depend on the data source.
  - **RESTRICT**: refuses to delete the data source if any objects depend on it. This is the default action.

    Currently, no objects depend on data sources. Therefore, **CASCADE** is equivalent to **RESTRICT**, and they are reserved to ensure backward compatibility.

Examples

```
-- Create a data source.
openGauss=# CREATE DATA SOURCE ds_tst1;

-- Delete the data source.
openGauss=# DROP DATA SOURCE ds_tst1 CASCADE;
openGauss=# DROP DATA SOURCE IF EXISTS ds_tst1 RESTRICT;
```

Helpful Links

CREATE DATA SOURCE and **ALTER DATA SOURCE**

11.14.111 DROP DIRECTORY

Function

DROP Directory deletes a synonym.

Precautions

When **enable_access_server_directory** is set to **off**, only the initial user is allowed to delete directory objects. When **enable_access_server_directory** is set to **on**, a user with the SYSADMIN permission, the owner of the directory object, a user who is granted with the DROP permission of the directory, or a user who inherits the **gs_role_directory_drop** permission of the built-in role can delete directory objects.

Syntax

```
DROP DIRECTORY [ IF EXISTS ] directory_name;
```

Parameter Description

- **directory_name**

  Specifies the name of the directory to be deleted.

  Value range: an existing directory name

Examples

```
-- Create a directory.
openGauss=# CREATE OR REPLACE DIRECTORY  dir  as '/tmp/';

-- Delete a directory.
openGauss=# DROP DIRECTORY dir;
```

Helpful Links

CREATE DIRECTORY and **ALTER DIRECTORY**

11.14.112 DROP EXTENSION

Function

DROP EXTENSION deletes an extension.

Precautions

- **DROP EXTENSION** deletes an extension from the database. When you delete an extension, the components that make up the extension are also deleted.
- The **DROP EXTENSION** command can be used only by the owner of the extension.

Syntax

```
DROP EXTENSION [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

Parameter Description

- **IF EXISTS**

  If the **IF EXISTS** parameter is used and the extension does not exist, no error is reported. Instead, a **NOTICE** is generated.
- **name**

  Name of an installed extension.
- **CASCADE**

  Automatically deletes objects that depend on the extension.
- **RESTRICT**

  If any object depends on an extension, the extension cannot be deleted (unless all its member objects and other extension objects are deleted at a time using the **DROP** command). This is a default processing.

Examples

Delete extension **hstore** from the current database.

```
DROP EXTENSION hstore;
```

In the current database, if an object that uses **hstore** exists, this command will fail. For example, a column in any table is of the **hstore** type. Adding the **CASCADE** option can forcibly delete the extension and objects that depend on it.

11.14.113 DROP FOREIGN TABLE

Function

DROP FOREIGN TABLE drops a foreign table.

Precautions

DROP FOREIGN TABLE forcibly drops the specified table and the indexes depending on the table. After the table is dropped, the functions and stored procedures that need to use this table cannot be executed.

Syntax

```
DROP FOREIGN TABLE [ IF EXISTS ]
    table_name [, ...] [ CASCADE | RESTRICT ];
```

Parameter Description

- **IF EXISTS**

  Reports a notice instead of an error if the specified table does not exist.
- **table_name**

  Table name

  Value range: an existing table name
- **CASCADE | RESTRICT**
  - **CASCADE**: automatically drops the objects (such as views) that depend on the table.
  - **RESTRICT**: refuses to drop the table if any objects depend on it. This is the default action.

Helpful Links

ALTER FOREIGN TABLE and **CREATE FOREIGN TABLE**

11.14.114 DROP FUNCTION

Function

DROP FUNCTION deletes a function.

Precautions

- If a function involves operations on temporary tables, **DROP FUNCTION** cannot be used.
- Only the function owner or a user granted with the DROP permission can run the **DROP FUNCTION** command. The system administrator has this permission by default.

Syntax

```
DROP FUNCTION [ IF EXISTS ] function_name
[ ( [ {[ argname ] [ argmode ] argtype} [, ...] ] ) [ CASCADE | RESTRICT ] ];
```

Parameter Description

- **IF EXISTS**

  Reports a notice instead of an error if the specified function does not exist.
- **function_name**

Specifies the name of the function to be deleted.

Value range: an existing function name

- **argmode**

  Specifies the parameter mode of the function.
- **argname**

  Specifies the parameter name of the function.
- **argtype**

  Specifies the parameter type of the function.

Examples

For details, see **Examples**.

Helpful Links

ALTER FUNCTION and **CREATE FUNCTION**

11.14.115 DROP GLOBAL CONFIGURATION

Function

DROP GLOBAL CONFIGURATION deletes parameter values from the **gs_global_config** system catalog.

Precautions

- Only the initial database user can run this command.
- The **weak_password** keyword cannot be deleted.

Syntax

```
DROP GLOBAL CONFIGURATION Parameter name, Parameter name...;
```

Parameter Description

The parameter is a parameter that already exists in the **gs_global_config** system catalog. If you delete a parameter that does not exist, an error will be reported.

11.14.116 DROP GROUP

Function

DROP GROUP deletes a user group.

DROP GROUP is an alias for **DROP ROLE**.

Precautions

DROP GROUP is an interface of the GaussDB management tool. You are not advised to use this interface, because doing so affects GaussDB.

Syntax

```
DROP GROUP [ IF EXISTS ] group_name [, ...];
```

Parameter Description

See **Parameter Description** in **DROP ROLE**.

Helpful Links

CREATE GROUP, **ALTER GROUP**, and **DROP ROLE**

11.14.117 DROP INDEX

Function

DROP INDEX deletes an index.

Precautions

Only the index owner, a user of a schema where the index resides, or a user who has the INDEX permission on the table where the index resides can run the **DROP INDEX** command. The system administrator has this permission by default.

For a global temporary table, if a session has initialized a global temporary table object (including creating a global temporary table and inserting data into the global temporary table for the first time), other sessions cannot delete indexes from the table.

Syntax

```
DROP INDEX [ CONCURRENTLY ] [ IF EXISTS ]
    index_name [, ...] [ CASCADE | RESTRICT ];
```

Parameter Description

- **CONCURRENTLY**

  Deletes an index without locking it. A normal **DROP INDEX** acquires exclusive lock on the table on which the index depends, blocking other accesses until the index drop can be completed. With this option, the statement does not lock the table during index deletion.

  This parameter allows only one index name and does not support **CASCADE**.

  The **DROP INDEX** statement can be run within a transaction, but **DROP INDEX CONCURRENTLY** cannot.

- **IF EXISTS**

  Reports a notice instead of an error if the specified index does not exist.

- **index_name**

  Specifies the name of the index to be deleted.

  Value range: an existing index

- **CASCADE | RESTRICT**
  - **CASCADE**: automatically deletes the objects that depend on the index.

– **RESTRICT**: refuses to delete the index if any objects depend on it. This is the default action.

Examples

See **Examples** in **CREATE INDEX**.

Helpful Links

ALTER INDEX and **CREATE INDEX**.

11.14.118 DROP LANGUAGE

This version does not support this syntax.

11.14.119 DROP MASKING POLICY

Function

DROP MASKING POLICY deletes an anonymization policy.

Precautions

Only user **poladmin**, user **sysadmin**, or the initial user can perform this operation.

Syntax

```
DROP MASKING POLICY [IF EXISTS] policy_name;
```

Parameter Description

policy_name

Specifies the audit policy name, which must be unique.

Value range: a string. It must comply with the naming convention.

Examples

```
-- Delete a masking policy.
openGauss=# DROP MASKING POLICY IF EXISTS maskpol1;

-- Delete a group of masking policies.
openGauss=# DROP MASKING POLICY IF EXISTS maskpol1, maskpol2, maskpol3;
```

Helpful Links

ALTER MASKING POLICY and **CREATE MASKING POLICY**

11.14.120 DROP MATERIALIZED VIEW

Function

DROP MATERIALIZED VIEW forcibly deletes an existing materialized view from the database.

Precautions

The owner of a materialized view, owner of the schema of the materialized view, users granted with the DROP permission on the materialized view, or users granted with the DROP ANY TABLE permission can run the **DROP MATERIALIZED VIEW** command. By default, the system administrator has the permission to run the command.

Syntax

```
DROP MATERIALIZED VIEW [ IF EXISTS ] mv_name [, ...] [ CASCADE | RESTRICT ];
```

Parameter Description

- **IF EXISTS**

  Reports a notice instead of an error if the specified materialized view does not exist.

- **mv_name**

  Name of the materialized view to be deleted.

- **CASCADE | RESTRICT**
  - **CASCADE**: automatically deletes the objects that depend on the materialized view.
  - **RESTRICT**: refuses to delete the materialized view if any objects depend on it. This is the default action.

Examples

```
-- Delete the materialized view named my_mv.
openGauss=# DROP MATERIALIZED VIEW my_mv;
```

Helpful Links

ALTER MATERIALIZED VIEW, **CREATE INCREMENTAL MATERIALIZED VIEW**, **CREATE MATERIALIZED VIEW**, **CREATE TABLE**, **REFRESH INCREMENTAL MATERIALIZED VIEW**, and **REFRESH MATERIALIZED VIEW**

11.14.121 DROP MODEL

Function

DROP MODEL deletes a model object that has been trained and saved.

Precautions

The deleted model can be viewed in the **gs_model_warehouse** system catalog.

Syntax

```
DROP MODEL model_name;
```

Parameter Description

model_name

Specifies a model name.

Value range: a string. It must comply with the identifier naming convention.

Helpful Links

CREATE MODEL and **PREDICT BY**

11.14.122 DROP OPERATOR

Centralized systems do not support the DROP OPERATOR function.

11.14.123 DROP OWNED

Function

DROP OWNED deletes the database objects owned by a database role.

Precautions

- This interface will revoke the role's permissions on all objects in the current database and shared objects (databases and tablespaces).
- **DROP OWNED** is often used to prepare for removing one or more roles. Because **DROP OWNED** affects only the objects in the current database, you need to run this statement in each database that contains the objects owned by the role to be removed.
- Using the **CASCADE** option may cause this statement to recursively remove objects owned by other users.
- The databases and tablespaces owned by the role will not be removed.

Syntax

```
DROP OWNED BY name [, ...] [ CASCADE | RESTRICT ];
```

Parameter Description

- **name**

  Specifies the role name.
- **CASCADE | RESTRICT**
  - **CASCADE**: automatically deletes the objects that depend on the objects to be deleted.
  - **RESTRICT**: refuses to delete the objects if other objects depend on them. This is the default action.

Helpful Links

REASSIGN OWNED and **DROP ROLE**

11.14.124 DROP PACKAGE

Function

DROP PACKAGE deletes the existing package or package body.

Precautions

After the package body is deleted, the stored procedures and functions in the package become invalid at the same time.

Syntax

```
DROP PACKAGE [ IF EXISTS ] package_name;
DROP PACKAGE BODY [ IF EXISTS ] package_name;
```

11.14.125 DROP PROCEDURE

Function

DROP PROCEDURE deletes a stored procedure.

Precautions

None

Syntax

```
DROP PROCEDURE [ IF EXISTS  ] procedure_name ;
```

Parameter Description

- **IF EXISTS**

  Reports a notice instead of an error if the specified stored procedure does not exist.

- **procedure_name**

  Specifies the name of the stored procedure to be deleted.

  Value range: an existing stored procedure name

Helpful Links

CREATE PROCEDURE

11.14.126 DROP RESOURCE LABEL

Function

DROP RESOURCE LABEL deletes a resource label.

Precautions

Only users with the **poladmin** or **sysadmin** permission, or the initial user can perform this operation.

Syntax

```
DROP RESOURCE LABEL [IF EXISTS] policy_name[, ...]*;
```

Parameter Description

label_name

Specifies the resource label name.

Value range: a string. It must comply with the naming convention.

Examples

```
-- Delete a resource label.
openGauss=# DROP RESOURCE LABEL IF EXISTS res_label1;

-- Delete a group resource label.
openGauss=# DROP RESOURCE LABEL IF EXISTS res_label1, res_label2, res_label3;
```

Helpful Links

ALTER RESOURCE LABEL and **CREATE RESOURCE LABEL**

11.14.127 DROP ROLE

Function

DROP ROLE deletes a role.

Precautions

None

Syntax

```
DROP ROLE [ IF EXISTS ] role_name [, ...];
```

Parameter Description

- **IF EXISTS**
  Reports a notice instead of an error if the specified role does not exist.
- **role_name**
  Specifies the name of the role to be deleted.
  Value range: an existing role name

Examples

See **Examples** in **CREATE ROLE**.

Helpful Links

CREATE ROLE, **ALTER ROLE**, and **SET ROLE**

11.14.128 DROP ROW LEVEL SECURITY POLICY

Function

DROP ROW LEVEL SECURITY POLICY deletes a row-level access control policy from a table.

Precautions

Only the table owner or administrators can delete a row-level access control policy from the table.

Syntax

```
DROP [ ROW LEVEL SECURITY ] POLICY [ IF EXISTS ] policy_name ON table_name [ CASCADE | RESTRICT ]
```

Parameter Description

- **IF EXISTS**

  Reports a notice instead of an error if the specified row-level access control policy does not exist.

- **policy_name**

  Specifies the name of the row-level access control policy to be deleted.

  – table_name

    Specifies the name of the table containing the row-level access control policy.

  – CASCADE/RESTRICT

    Currently, no objects depend on row-level access control policies. Therefore, **CASCADE** is equivalent to **RESTRICT**, and they are reserved to ensure backward compatibility.

Examples

```
-- Create the data table all_data.
openGauss=# CREATE TABLE all_data(id int, role varchar(100), data varchar(100));

-- Create a row-level access control policy.
openGauss=# CREATE ROW LEVEL SECURITY POLICY all_data_rls ON all_data USING(role =
CURRENT_USER);

-- Delete a row-level access control policy.
openGauss=# DROP ROW LEVEL SECURITY POLICY all_data_rls ON all_data;
```

Helpful Links

ALTER ROW LEVEL SECURITY POLICY and **CREATE ROW LEVEL SECURITY POLICY**

11.14.129 DROP RULE

Function

DROP RULE deletes a rewriting rule.

Syntax

```
DROP RULE [ IF EXISTS ] name ON table_name [ CASCADE | RESTRICT ]
```

Parameter Description

- IF EXISTS
  If the rule does not exist, a **NOTICE** is thrown.
- name
  Name of an existing rule to be deleted.
- table_name
  Name of the table to which the rule applies.
- CASCADE
  Automatically cascade deletes objects that depend on this rule.
- RESTRICT
  By default, if any objects depend on the rule, the rule cannot be deleted.

Examples

```
-- Delete a rewriting rule (newrule).
DROP RULE newrule ON mytable;
```

11.14.130 DROP PUBLICATION

Function

DROP PUBLICATION deletes an existing publication from a database.

Precautions

A publication can be deleted only by its owner or the system administrator.

Syntax

```
DROP PUBLICATION [ IF EXISTS ] name [ CASCADE | RESTRICT ]
```

Parameter Description

- **IF EXISTS**
  Does not throw an error if a publication does not exist, but instead reports a notice.
- **name**
  Specifies the name of an existing publication.
- **CASCADE|RESTRICT**

Currently, these keywords do not work because there is no dependency on publications.

Examples

For details, see **Examples**.

Helpful Links

ALTER PUBLICATION and **CREATE PUBLICATION**

11.14.131 DROP SCHEMA

Function

DROP SCHEMA deletes a schema from the current database.

Precautions

Only the schema owner or a user granted with the DROP permission can run the **DROP SCHEMA** command. The system administrator has this permission by default.

Syntax

```
DROP SCHEMA [ IF EXISTS ] schema_name [, ...] [ CASCADE | RESTRICT ];
```

Parameter Description

- **IF EXISTS**

  Reports a notice instead of an error if the specified schema does not exist.
- **schema_name**

  Specifies the name of the schema to be deleted.

  Value range: an existing schema name
- **CASCADE | RESTRICT**
  - **CASCADE**: automatically deletes all the objects contained in the schema.
  - **RESTRICT**: refuses to delete the schema if the schema contains objects. This is the default action.

NOTICE

Schemas beginning with **pg_temp** or **pg_toast_temp** are for internal use. Do not delete them. Otherwise, unexpected consequences may be incurred.

NOTE

The schema currently being used cannot be deleted. To delete it, switch to another schema first.

Examples

See **Examples** in **CREATE SCHEMA**.

Helpful Links

ALTER SCHEMA and **CREATE SCHEMA**

11.14.132 DROP SEQUENCE

Function

DROP SEQUENCE deletes a sequence from the current database.

Precautions

- Only the owner of a sequence, the owner of the schema of the sequence, or users granted with the DROP permission on the sequence can delete the sequence. By default, the system administrator has the permission to delete the sequence.
- If the LARGE identifier is used when a sequence is created, the LARGE identifier must be used when the sequence is dropped.

Syntax

```
DROP [ LARGE ] SEQUENCE [ IF EXISTS ] {[schema.]sequence_name} [ , ... ] [ CASCADE | RESTRICT ];
```

Parameter Description

- **IF EXISTS**

  Reports a notice instead of an error if the specified sequence does not exist.

- **name**

  Specifies the name of the sequence to be deleted.

- **CASCADE**

  Automatically deletes the objects that depend on the sequence.

- **RESTRICT**

  Refuses to delete the sequence if any objects depend on it. This is the default action.

Examples

```
-- Create an ascending sequence named serial, starting from 101.
openGauss=# CREATE SEQUENCE serial START 101;

-- Delete a sequence.
openGauss=# DROP SEQUENCE serial;
```

Helpful Links

ALTER SEQUENCE and **DROP SEQUENCE**

11.14.133 DROP SERVER

Function

DROP SERVER drops a data server.

Precautions

Only the server owner or a user granted with the DROP permission can run the **DROP SERVER** command. The system administrator has this permission by default.

Syntax

```
DROP SERVER [ IF EXISTS ] server_name [ {CASCADE | RESTRICT} ] ;
```

Parameter Description

- **IF EXISTS**

  Reports a notice instead of an error if the specified data server does not exist.
- **server_name**

  Specifies the name of the data server to be dropped.
- **CASCADE | RESTRICT**
  - **CASCADE**: automatically drops the objects that depend on the data server.
  - **RESTRICT**: refuses to drop the server if any objects depend on it. This is the default action.

Helpful Links

ALTER SERVER and **CREATE SERVER**

11.14.134 DROP SUBSCRIPTION

Function

DROP SUBSCRIPTION deletes a subscription from a database instance.

Precautions

- A subscription can be deleted only by the system administrator.
- If the subscription to be deleted is associated with a replication slot, **DROP SUBSCRIPTION** cannot be executed inside a transaction block.

Syntax

```
DROP SUBSCRIPTION [ IF EXISTS ] name [ CASCADE | RESTRICT ]
```

Parameter Description

- **name**

  Specifies the name of the subscription to be deleted.

- **CASCADE|RESTRICT**

  Currently, these keywords do not work because there is no dependency on subscriptions.

Examples

For details, see **Examples**.

Helpful Links

ALTER SUBSCRIPTION and **CREATE SUBSCRIPTION**

11.14.135 DROP SYNONYM

Function

DROP SYNONYM deletes a synonym.

Precautions

Only the owner of a synonym or a system administrator has the **DROP SYNONYM** permission.

Syntax

```
DROP SYNONYM [ IF EXISTS ] synonym_name [ CASCADE | RESTRICT ];
```

Parameter Description

- **IF EXISTS**

  Reports a notice instead of an error if the specified synonym does not exist.

- **synonym_name**

  Specifies the name (optionally schema-qualified) of the synonym to be deleted.

- **CASCADE | RESTRICT**
  - **CASCADE**: automatically deletes the objects (such as views) that depend on the synonym.
  - **RESTRICT**: refuses to delete the synonym if any objects depend on it. This is the default action.

Examples

See **Examples** in **CREATE SYNONYM**.

Helpful Links

ALTER SYNONYM and **CREATE SYNONYM**

11.14.136 DROP TABLE

Function

DROP TABLE deletes a table.

Precautions

- **DROP TABLE** forcibly deletes the specified table and the indexes depending on the table. After the table is deleted, the functions and stored procedures that need to use this table cannot be executed. Deleting a partitioned table also deletes all partitions in the table.
- The owner of a table, the owner of the schema of the table, users granted with the DROP permission on the table, or users granted with the DROP ANY TABLE permission can delete the specified table. The system administrator has the permission to delete the specified table by default.

Syntax

```
DROP TABLE [ IF EXISTS ]
    { [schema.]table_name } [, ...] [ CASCADE | RESTRICT ] [ PURGE ];
```

Parameter Description

- **IF EXISTS**

  Reports a notice instead of an error if the specified table does not exist.
- **schema**

  Specifies the schema name.
- **table_name**

  Specifies the name of the table to be deleted.
- **CASCADE | RESTRICT**
  - **CASCADE**: automatically deletes the objects (such as views) that depend on the table.
  - **RESTRICT**: refuses to delete the table if any objects depend on it. This is the default action.
- **PURGE**

  Specifies that even if the recycle bin function is enabled, the table is physically dropped instead of being moved to the recycle bin.

Examples

See **Examples** in **CREATE TABLE**.

Helpful Links

ALTER TABLE and **CREATE TABLE**

11.14.137 DROP TABLESPACE

Function

DROP TABLESPACE deletes a tablespace.

Precautions

- Only the tablespace owner or a user granted with the DROP permission can run the **DROP TABLESPACE** command. The system administrator has this permission by default.
- The tablespace to be deleted should not contain any database objects. Otherwise, an error will be reported.
- **DROP TABLESPACE** cannot be rolled back and therefore cannot be run in transaction blocks.
- During execution of **DROP TABLESPACE**, database queries by other sessions using **\db** may fail and need to be reattempted.
- If **DROP TABLESPACE** fails to be executed, run **DROP TABLESPACE IF EXISTS**.

Syntax

```
DROP TABLESPACE [ IF EXISTS ] tablespace_name;
```

Parameter Description

- **IF EXISTS**

  Reports a notice instead of an error if the specified tablespace does not exist.
- **tablespace_name**

  Specifies the name of the tablespace to be deleted.

  Value range: an existing tablespace name

Examples

See **Examples** in **CREATE TABLESPACE**.

Helpful Links

ALTER TABLESPACE and **CREATE TABLESPACE**

Suggestions

- drop tablespace

  Do not delete tablespaces during transactions.

11.14.138 DROP TEXT SEARCH CONFIGURATION

Function

DROP TEXT SEARCH CONFIGURATION deletes a text search configuration.

Precautions

Only the owner of a text search configuration has the **DROP TEXT SEARCH CONFIGURATION** permission.

Syntax

```
DROP TEXT SEARCH CONFIGURATION [ IF EXISTS ] name [ CASCADE | RESTRICT ];
```

Parameter Description

- **IF EXISTS**

  Reports a notice instead of an error if the specified text search configuration does not exist.
- **name**

  Specifies the name (optionally schema-qualified) of the text search configuration to be deleted.
- **CASCADE**

  Automatically deletes the objects that depend on the text search configuration.
- **RESTRICT**

  Refuses to delete the text search configuration if any objects depend on it. This is the default action.

Examples

See **Examples** in **CREATE TEXT SEARCH CONFIGURATION**.

Helpful Links

ALTER TEXT SEARCH CONFIGURATION and **CREATE TEXT SEARCH CONFIGURATION**

11.14.139 DROP TEXT SEARCH DICTIONARY

Function

DROP TEXT SEARCH DICTIONARY deletes a full-text retrieval dictionary.

Precautions

- Predefined dictionaries do not support the **DROP** operation.
- Only the owner of a dictionary or a system administrator has the permission to **DROP**.
- Execute **DROP...CASCADE** only when necessary because this operation will delete the text search configurations that use this dictionary.

Syntax

```
DROP TEXT SEARCH DICTIONARY [ IF EXISTS ] name [ CASCADE | RESTRICT ]
```

Parameter Description

- **IF EXISTS**

  Reports a notice instead of an error if the specified full-text retrieval dictionary does not exist.
- **name**

  Specifies the name (optionally schema-qualified) of the full-text retrieval dictionary to be deleted. (If you do not specify a schema name, the dictionary will be deleted in the current schema by default.)

  Value range: an existing dictionary name
- **CASCADE**

  Automatically deletes the objects that depend on the full-text retrieval dictionary and other objects that depend on these objects.

  If any text search configuration uses the dictionary, the **DROP** statement will fail. You can add **CASCADE** to delete all text search configurations and dictionaries that use this dictionary.
- **RESTRICT**

  Refuses to delete the full-text retrieval dictionary if any object depends on it. It is the default value.

Examples

```
-- Delete the english dictionary.
openGauss=# DROP TEXT SEARCH DICTIONARY english;
```

Helpful Links

ALTER TEXT SEARCH DICTIONARY and CREATE TEXT SEARCH DICTIONARY

11.14.140 DROP TRIGGER

Function

DROP TRIGGER deletes a trigger.

Precautions

Only the owner of a trigger or a system administrator has the **DROP TRIGGER** permission.

Syntax

```
DROP TRIGGER [ IF EXISTS ] trigger_name ON table_name [ CASCADE | RESTRICT ];
```

Parameter Description

- **IF EXISTS**

  Reports a notice instead of an error if the specified trigger does not exist.
- **trigger_name**

  Specifies the name of the trigger to be deleted.

  Value range: an existing trigger name

- **table_name**

  Specifies the name of the table containing the trigger.

  Value range: name of the table containing the trigger

- **CASCADE | RESTRICT**
  - **CASCADE**: automatically deletes the objects that depend on the trigger.
  - **RESTRICT**: refuses to delete the trigger if any objects depend on it. This is the default action.

Examples

For details, see **Examples** in **CREATE TRIGGER**.

Helpful Links

CREATE TRIGGER, **ALTER TRIGGER**, and **ALTER TABLE**

11.14.141 DROP TYPE

Function

DROP TYPE deletes a user-defined data type.

Precautions

Only the type owner or a user granted with the DROP permission can run the **DROP TYPE** command. The system administrator has this permission by default.

Syntax

```
DROP TYPE [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

Parameter Description

- **IF EXISTS**

  Reports a notice instead of an error if the specified type does not exist.

- **name**

  Specifies the name (optionally schema-qualified) of the type to be deleted.

- **CASCADE**

  Automatically deletes the objects (such as fields, functions, and operators) that depend on the type.

  RESTRICT

  Refuses to delete the type if any objects depend on it. This is the default action.

Examples

See **Examples** in **CREATE TYPE**.

Helpful Links

CREATE TYPE and **ALTER TYPE**

11.14.142 DROP USER

Function

DROP USER deletes a user and the schema with the same name as the user.

Precautions

- **CASCADE** is used to delete the objects (excluding databases) that depend on the user. **CASCADE** cannot delete locked objects unless the objects are unlocked or the processes locking the objects are killed.
- In GaussDB, the **enable_kill_query** configuration parameter exists in the **postgresql.conf** file. This parameter affects **CASCADE**.
  - If **enable_kill_query** is **on** and **CASCADE** is used, the statement automatically kills the processes locking dependent objects and then deletes the specified user.
  - If **enable_kill_query** is **off** and **CASCADE** is used, the statement waits until the processes locking dependent objects stop and then deletes the specified user.
- If the dependent objects are other databases or reside in other databases, manually delete them before deleting the user from the current database. **DROP USER** cannot delete objects across databases.
- Before deleting a user, you need to delete all the objects owned by the user and revoke the user's permissions on other objects. Alternatively, you can specify CASCADE to delete the objects owned by the user and the granted permissions.
- If a data source depends on the user, the user cannot be deleted directly. You need to manually delete the data source first.

Syntax

```
DROP USER [ IF EXISTS ] user_name [, ...] [ CASCADE | RESTRICT ];
```

Parameter Description

- **IF EXISTS**

  Reports a notice instead of an error if the specified user does not exist.
- **user_name**

  Specifies the name of the user to be deleted.

  Value range: an existing username
- **CASCADE | RESTRICT**
  - **CASCADE**: automatically deletes objects that depend on the user and revokes the permissions granted to the user.
  - **RESTRICT**: refuses to delete a user if the user has any dependent objects or has been granted permissions on other objects. This is the default value.

NOTE

In GaussDB, the **enable_kill_query** configuration parameter exists in the **postgresql.conf** file. This parameter affects **CASCADE**.

- If **enable_kill_query** is **on** and **CASCADE** is used, the statement automatically kills the processes locking dependent objects and then deletes the specified user.
- If **enable_kill_query** is **off** and **CASCADE** is used, the statement waits until the processes locking dependent objects stop and then deletes the specified user.

Examples

See **Examples** in **CREATE USER**.

Helpful Links

ALTER USER and **CREATE USER**.

11.14.143 DROP USER MAPPING

Function

DROP USER MAPPING drops a user mapping for a foreign server.

Syntax

```
DROP USER MAPPING [ IF EXISTS ] FOR { user_name | USER | CURRENT_USER | PUBLIC } SERVER
server_name;
```

Parameter Description

- **IF EXISTS**

  Reports a notice instead of an error if the user mapping does not exist.
- **user_name**

  Specifies user name of the mapping.

  CURRENT_USER and USER match the name of the current user. PUBLIC is used to match all current and future user names in the system.
- **server_name**

  Specifies name of the server to which the user is mapped.

Helpful Links

ALTER USER MAPPING and **CREATE USER MAPPING**

11.14.144 DROP VIEW

Function

DROP VIEW forcibly deletes a view from the database.

Precautions

The owner of a view, owner of the schema of the view, users granted with the DROP permission on the view, or users granted with the DROP ANY TABLE permission can run the **DROP VIEW** command. By default, the system administrator has the permission to run the command.

Syntax

```
DROP VIEW [ IF EXISTS ] view_name [, ...] [ CASCADE | RESTRICT ];
```

Parameter Description

- **IF EXISTS**

  Reports a notice instead of an error if the specified view does not exist.
- **view_name**

  Specifies the name of the view to be deleted.

  Value range: an existing view name
- **CASCADE | RESTRICT**
  - **CASCADE**: automatically deletes the objects (such as other views) that depend on the view.
  - **RESTRICT**: refuses to delete the view if any objects depend on it. This is the default action.

Examples

See **Examples** in **CREATE VIEW**.

Helpful Links

ALTER VIEW and **CREATE VIEW**

11.14.145 DROP WEAK PASSWORD DICTIONARY

Function

DROP WEAK PASSWORD DICTIONARY clears all weak passwords in **gs_global_config**.

Precautions

Only the initial user, system administrator, and security administrator have the permission to execute this syntax.

Syntax

```
DROP WEAK PASSWORD DICTIONARY;
```

Parameter Description

None

Examples

See the examples in **CREATE WEAK PASSWORD DICTIONARY**.

Helpful Links

CREATE WEAK PASSWORD DICTIONARY

11.14.146 EXECUTE

Function

EXECUTE executes a prepared statement. Because a prepared statement exists only in the lifetime of the session, the prepared statement must be created earlier in the current session by using the **PREPARE** statement.

Precautions

If the **PREPARE** statement creating the prepared statement declares some parameters, the parameter set passed to the **EXECUTE** statement must be compatible. Otherwise, an error will occur.

Syntax

```
EXECUTE name [ ( parameter [, ...] ) ];
```

Parameter Description

- **name**

  Specifies the name of the prepared statement to be executed.

- **parameter**

  Specifies a parameter of the prepared statement. It must be of the same data type as that specified parameter in creating and generating the prepared statement.

Examples

```
-- Create the reason table.
openGauss=# CREATE TABLE tpcds.reason (
    CD_DEMO_SK          INTEGER          NOT NULL,
    CD_GENDER           character(16)            ,
    CD_MARITAL_STATUS   character(100)
)
;

-- Insert data.
openGauss=# INSERT INTO tpcds.reason VALUES(51, 'AAAAAAAADDAAAAAA', 'reason 51');

-- Create the reason_t1 table.
openGauss=# CREATE TABLE tpcds.reason_t1 AS TABLE tpcds.reason;

-- Create a prepared statement for an INSERT statement and execute the prepared statement.
openGauss=# PREPARE insert_reason(integer,character(16),character(100)) AS INSERT INTO
tpcds.reason_t1 VALUES($1,$2,$3);

openGauss=# EXECUTE insert_reason(52, 'AAAAAAAADDAAAAAA', 'reason 52');

-- Delete the reason and reason_t1 tables.
```

```
openGauss=# DROP TABLE tpcds.reason;
openGauss=# DROP TABLE tpcds.reason_t1;
```

11.14.147 EXPLAIN

Function

EXPLAIN shows the execution plan of an SQL statement.

The execution plan shows how the tables referenced by the statement will be scanned - by plain sequential scan, index scan, etc. - and if multiple tables are referenced, what join algorithms will be used to bring together the required rows from each input table.

The most critical part of the display is the estimated statement execution cost, which is the planner's guess at how long it will take to run the statement.

The **ANALYZE** option causes the statement to be actually executed, not only planned. The total elapsed time expended within each plan node (in milliseconds) and total number of rows it actually returned are added to the display. This is useful for seeing whether the planner's estimates are close to reality.

Precautions

- The statement is actually executed when the **ANALYZE** option is used. If you want to use **EXPLAIN** to analyze **INSERT**, **UPDATE**, **DELETE**, **CREATE TABLE AS**, or **EXECUTE** statement without letting the statement affect your data, use this approach:
```
START TRANSACTION;
EXPLAIN ANALYZE ...;
ROLLBACK;
```
- The **DETAIL**, **NODES**, and **NUM_NODES** parameters are disabled in standalone mode. They are available only in distributed mode. If the parameters are used, the following error is reported:
```
openGauss=# create table student(id int, name char(20));
CREATE TABLE
openGauss=# explain (nodes true) insert into student values(5,'a'),(6,'b');
ERROR:  unrecognized EXPLAIN option "nodes"
openGauss=# explain (num_nodes true) insert into student values(5,'a'),(6,'b');
ERROR:  unrecognized EXPLAIN option "num_nodes"
```

Syntax

- Display the execution plan of an SQL statement, which supports multiple options and has no requirements for the order of options.
```
EXPLAIN [ (  option  [, ...] )  ] statement;
```
The syntax of the **option** clause is as follows:
```
ANALYZE [ boolean ] |
    ANALYSE [ boolean ] |
    VERBOSE [ boolean ] |
    COSTS [ boolean ] |
    CPU [ boolean ] |
    DETAIL [ boolean ] | (available only in distributed mode))
    NODES [ boolean ] | (available only in distributed mode)
    NUM_NODES [ boolean ] | (available only in distributed mode)
    BUFFERS [ boolean ] |
    TIMING [ boolean ] |
    PLAN [ boolean ] |
    FORMAT { TEXT | XML | JSON | YAML }
```

- Display the execution plan of an SQL statement, where options are in order.

```
EXPLAIN { [ { ANALYZE | ANALYSE } ] [ VERBOSE ] | PERFORMANCE } statement;
```

Parameter Description

- **statement**

  Specifies the SQL statement to explain.

- **ANALYZE boolean | ANALYSE boolean**

  Specifies whether to display actual run times and other statistics.

  Value range:

  – **TRUE** (default): displays them.

  – **FALSE**: does not display them.

- **VERBOSE boolean**

  Specifies whether to display additional information regarding the plan.

  Value range:

  – **TRUE** (default): displays it.

  – **FALSE**: does not display it.

- **COSTS boolean**

  Specifies whether to display the estimated total cost of each plan node, estimated number of rows, estimated width of each row.

  Value range:

  – **TRUE** (default): displays them.

  – **FALSE**: does not display them.

- **CPU boolean**

  Specifies whether to display CPU usage.

  Value range:

  – **TRUE** (default): displays it.

  – **FALSE**: does not display it.

- **DETAIL boolean** (available only in distributed mode)

  Displays information about database nodes.

  Value range:

  – **TRUE** (default): displays it.

  – **FALSE**: does not display it.

- **NODES boolean** (available only in distributed mode)

  Specifies whether to display information about the nodes executed by query.

  Value range:

  – **TRUE** (default): displays it.

  – **FALSE**: does not display it.

- **NUM_NODES boolean** (available only in distributed mode)

  Specifies whether to display the number of executing nodes.

  Value range:

  – **TRUE** (default): displays it.

  – **FALSE**: does not display it.

- **BUFFERS boolean**

  Specifies whether to display buffer usage.

  Value range:

  – **TRUE**: displays it.

  – **FALSE** (default): does not display it.

- **TIMING boolean**

  Specifies whether to display the actual startup time and time spent on the output node.

  Value range:

  – **TRUE** (default): displays them.

  – **FALSE**: does not display them.

- **PLAN**

  Specifies whether to store the execution plan in **PLAN_TABLE**. If this parameter is set to **on**, the execution plan is stored in **PLAN_TABLE** and not displayed on the screen. Therefore, this parameter cannot be used together with other parameters when it is set to **on**.

  Value range:

  – **ON** (default): The execution plan is stored in **PLAN_TABLE** and not printed on the screen. If the plan is stored successfully, "EXPLAIN SUCCESS" is returned.

  – **OFF**: The execution plan is not stored in **PLAN_TABLE** but is printed on the screen.

- **FORMAT**

  Specifies the output format.

  Value range: **TEXT**, **XML**, **JSON**, and **YAML**

  Default value: **TEXT**

- **PERFORMANCE**

  Prints all relevant information in execution.

Examples

```
-- Create the tpcds.customer_address_p1 table.
openGauss=# CREATE TABLE tpcds.customer_address_p1 AS TABLE tpcds.customer_address;

-- Change the value of explain_perf_mode to normal.
openGauss=# SET explain_perf_mode=normal;

-- Display an execution plan for simple queries in the table.
openGauss=# EXPLAIN SELECT * FROM tpcds.customer_address_p1;
QUERY PLAN
--------------------------------------------------
Data Node Scan  (cost=0.00..0.00 rows=0 width=0)
Node/s: All dbnodes
(2 rows)

-- Generate an execution plan in JSON format (with explain_perf_mode being normal).
openGauss=# EXPLAIN(FORMAT JSON) SELECT * FROM tpcds.customer_address_p1;
            QUERY PLAN
--------------------------------------
 [                              +
   {                            +
    "Plan": {                        +
      "Node Type": "Data Node Scan",+
      "Startup Cost": 0.00,         +
      "Total Cost": 0.00,           +
      "Plan Rows": 0,               +
      "Plan Width": 0,              +
      "Node/s": "All dbnodes"       +
    }                               +
  }                                 +
 ]
(1 row)

-- If there is an index and we use a query with an indexable WHERE condition, EXPLAIN might show a different plan.
openGauss=# EXPLAIN SELECT * FROM tpcds.customer_address_p1 WHERE ca_address_sk=10000;
QUERY PLAN
--------------------------------------------------
Data Node Scan  (cost=0.00..0.00 rows=0 width=0)
Node/s: dn_6005_6006
(2 rows)

-- Generate an execution plan in YAML format (with explain_perf_mode being normal).
openGauss=# EXPLAIN(FORMAT YAML) SELECT * FROM tpcds.customer_address_p1 WHERE
ca_address_sk=10000;
           QUERY PLAN
---------------------------------
 - Plan:                        +
     Node Type: "Data Node Scan"+
     Startup Cost: 0.00         +
     Total Cost: 0.00           +
     Plan Rows: 0               +
     Plan Width: 0              +
     Node/s: "dn_6005_6006"
(1 row)

-- Here is an example of a query plan with cost estimates suppressed:
openGauss=# EXPLAIN(COSTS FALSE)SELECT * FROM tpcds.customer_address_p1 WHERE
ca_address_sk=10000;
       QUERY PLAN
------------------------
 Data Node Scan
   Node/s: dn_6005_6006
(2 rows)

-- Here is an example of a query plan for a query using an aggregate function:
openGauss=# EXPLAIN SELECT SUM(ca_address_sk) FROM tpcds.customer_address_p1 WHERE
ca_address_sk<10000;
                                      QUERY PLAN
---------------------------------------------------------------------------------------
 Aggregate  (cost=18.19..14.32 rows=1 width=4)
   ->  Streaming (type: GATHER)  (cost=18.19..14.32 rows=3 width=4)
         Node/s: All dbnodes
         ->  Aggregate  (cost=14.19..14.20 rows=3 width=4)
               ->  Seq Scan on customer_address_p1  (cost=0.00..14.18 rows=10 width=4)
                     Filter: (ca_address_sk < 10000)
(6 rows)


-- Create a level-2 partitioned table.
openGauss=# CREATE TABLE range_list
openGauss-# (
openGauss(#     month_code VARCHAR2 ( 30 ) NOT NULL ,
openGauss(#     dept_code  VARCHAR2 ( 30 ) NOT NULL ,
openGauss(#     user_no    VARCHAR2 ( 30 ) NOT NULL ,
openGauss(#     sales_amt  int
openGauss(# )
openGauss-# PARTITION BY RANGE (month_code) SUBPARTITION BY LIST (dept_code)
openGauss-# (
openGauss(#   PARTITION p_201901 VALUES LESS THAN( '201903' )
openGauss(#   (
openGauss(#     SUBPARTITION p_201901_a values ('1'),
openGauss(#     SUBPARTITION p_201901_b values ('2')
openGauss(#   ),
openGauss(#   PARTITION p_201902 VALUES LESS THAN( '201910' )
openGauss(#   (
openGauss(#     SUBPARTITION p_201902_a values ('1'),
openGauss(#     SUBPARTITION p_201902_b values ('2')
openGauss(#   )
openGauss(# );
CREATE TABLE

-- Run a query statement containing a level-2 partitioned table.
-- Iterations and Sub Iterations specifies the numbers of level-1 and level-2 partitions that are traversed, respectively.
-- Selected Partitions specifies which level-1 partitions are actually scanned. Selected Subpartitions (p:s) indicates that s level-2 partitions under the pth level-1 partition are actually scanned. If all level-2 partitions under the level-1 partition are scanned, the value of s is ALL.
openGauss=# EXPLAIN SELECT * FROM range_list WHERE dept_code = '1';
                              QUERY PLAN
--------------------------------------------------------------------------------
 Partition Iterator  (cost=0.00..13.81 rows=2 width=238)
   Iterations: 2, Sub Iterations: 2
   ->  Partitioned Seq Scan on range_list  (cost=0.00..13.81 rows=2 width=238)
         Filter: ((dept_code)::text = '1'::text)
         Selected Partitions:  1..2
         Selected Subpartitions:  1:1, 2:1
(6 rows)

-- Delete the tpcds.customer_address_p1 table.
openGauss=# DROP TABLE tpcds.customer_address_p1;
```

Helpful Links

ANALYZE | ANALYSE

11.14.148 EXPLAIN PLAN

Function

EXPLAIN PLAN saves information about an execution plan into the **PLAN_TABLE** table. Different from the **EXPLAIN** statement, **EXPLAIN PLAN** only saves plan information and does not print information on the screen.

Syntax

```
EXPLAIN PLAN
[ SET STATEMENT_ID = string ]
FOR statement ;
```

Parameter Description

- **PLAN**: saves plan information into **PLAN_TABLE**. If information is stored successfully, "EXPLAIN SUCCESS" is returned.
- **STATEMENT_ID**: tags each query. The tag information will be stored in **PLAN_TABLE**.

  NOTE

  If the **EXPLAIN PLAN** statement does not contain **SET STATEMENT_ID**, **STATEMENT_ID** is empty by default. In addition, the value of **STATEMENT_ID** cannot exceed 30 bytes. Otherwise, an error will be reported.

Precautions

- **EXPLAIN PLAN** cannot be executed on a database node.
- Plan information cannot be collected for SQL statements that failed to be executed.
- Data in **PLAN_TABLE** is in a session-level lifecycle. Sessions are isolated from users, and therefore users can only view the data of the current session and current user.

Example 1

You can perform the following steps to collect execution plans of SQL statements by running **EXPLAIN PLAN**:

Step 1 Run the **EXPLAIN PLAN** statement.

NOTE

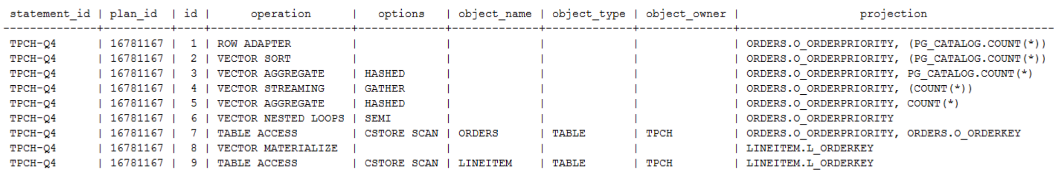
After the **EXPLAIN PLAN** statement is executed, plan information is automatically stored in **PLAN_TABLE**. **INSERT**, **UPDATE**, and **ANALYZE** cannot be performed on **PLAN_TABLE**.

For details about **PLAN_TABLE**, see **PLAN_TABLE**.

```
explain plan set statement_id='TPCH-Q4' for
select
o_orderpriority,
count(*) as order_count
from
orders
where
o_orderdate >= '1993-07-01'::date
and o_orderdate < '1993-07-01'::date + interval '3 month'
and exists (
select
*
from
lineitem
where
l_orderkey = o_orderkey
and l_commitdate < l_receiptdate
)
group by
o_orderpriority
order by
o_orderpriority;
```

Step 2 Query **PLAN_TABLE**.

```
SELECT * FROM PLAN_TABLE;
```



```
 statement_id | plan_id  | id |      operation       |   options    | object_name | object_type | object_owner |                 projection
--------------+----------+----+----------------------+--------------+-------------+-------------+--------------+---------------------------------------------
 TPCH-Q4      | 16781167 |  1 | ROW ADAPTER          |              |             |             |              | ORDERS.O_ORDERPRIORITY, (PG_CATALOG.COUNT(*))
 TPCH-Q4      | 16781167 |  2 | VECTOR SORT          |              |             |             |              | ORDERS.O_ORDERPRIORITY, (PG_CATALOG.COUNT(*))
 TPCH-Q4      | 16781167 |  3 | VECTOR AGGREGATE     | HASHED       |             |             |              | ORDERS.O_ORDERPRIORITY, PG_CATALOG.COUNT(*)
 TPCH-Q4      | 16781167 |  4 | VECTOR STREAMING     | GATHER       |             |             |              | ORDERS.O_ORDERPRIORITY, (COUNT(*))
 TPCH-Q4      | 16781167 |  5 | VECTOR AGGREGATE     | HASHED       |             |             |              | ORDERS.O_ORDERPRIORITY, COUNT(*)
 TPCH-Q4      | 16781167 |  6 | VECTOR NESTED LOOPS  | SEMI         |             |             |              | ORDERS.O_ORDERPRIORITY
 TPCH-Q4      | 16781167 |  7 | TABLE ACCESS         | CSTORE SCAN  | ORDERS      | TABLE       | TPCH         | ORDERS.O_ORDERPRIORITY, ORDERS.O_ORDERKEY
 TPCH-Q4      | 16781167 |  8 | VECTOR MATERIALIZE   |              |             |             |              | LINEITEM.L_ORDERKEY
 TPCH-Q4      | 16781167 |  9 | TABLE ACCESS         | CSTORE SCAN  | LINEITEM    | TABLE       | TPCH         | LINEITEM.L_ORDERKEY
```

Step 3 Delete data from **PLAN_TABLE**.

```
DELETE FROM PLAN_TABLE WHERE xxx;
```

----End

11.14.149 FETCH

Function

FETCH retrieves rows using a previously created cursor.

A cursor has an associated position, which is used by **FETCH**. The cursor position can be before the first row of the query result, on any particular row of the result, or after the last row of the result.

- When created, a cursor is positioned before the first row.
- After fetching some rows, the cursor is positioned on the row most recently retrieved.
- If **FETCH** runs off the end of the available rows then the cursor is left positioned after the last row, or before the first row if fetching backward.
- **FETCH ALL** or **FETCH BACKWARD ALL** will always leave the cursor positioned after the last row or before the first row.

Precautions

- If the cursor is declared with **NO SCROLL**, backward fetches like **FETCH BACKWARD** are not allowed.
- The forms **NEXT**, **PRIOR**, **FIRST**, **LAST**, **ABSOLUTE**, and **RELATIVE** fetch a single row after moving the cursor appropriately. If there is no such row, an empty result is returned, and the cursor is left positioned before the first row (backward fetch) or after the last row (forward fetch) as appropriate.
- The forms using **FORWARD** and **BACKWARD** retrieve the indicated number of rows moving in the forward or backward direction, leaving the cursor positioned on the last-returned row or after (backward fetch)/before (forward fetch) all rows if the **count** exceeds the number of rows available.
- **RELATIVE 0**, **FORWARD 0**, and **BACKWARD 0** all request fetching the current row without moving the cursor, that is, re-fetching the most recently fetched row. This will succeed unless the cursor is positioned before the first row or after the last row; in which case, no row is returned.
- If the cursor of **FETCH** involves a column-store table, backward fetches like **BACKWARD** and **PRIOR** are not allowed.

Syntax

```
FETCH [ direction { FROM | IN } ] cursor_name;
```

The **direction** clause specifies optional parameters.

```
NEXT
   | PRIOR
   | FIRST
   | LAST
   | ABSOLUTE count
   | RELATIVE count
   | count
   | ALL
   | FORWARD
   | FORWARD count
   | FORWARD ALL
   | BACKWARD
   | BACKWARD count
   | BACKWARD ALL
```

Parameter Description

- **direction_clause**

  Defines the fetch direction.

  Value range:

  – **NEXT** (default value)

    Fetches the next row.

  – PRIOR

    Fetches the prior row.

  – FIRST

    Fetches the first row of the query (same as **ABSOLUTE 1**).

  – LAST

    Fetches the last row of the query (same as **ABSOLUTE -1**).

  – ABSOLUTE count

    Fetches the *count*'th row of the query.

    ABSOLUTE fetches are not any faster than navigating to the desired row with a relative move: the underlying implementation must traverse all the intermediate rows anyway.

    Value range: a possibly-signed integer

    ■ If *count* is positive, the *count*'th row of the query will be fetched.

    ■ If *count* is negative, the **abs(***count***)**th row from the end of the query result will be fetched.

    ■ If *count* is set to **0**, the cursor is positioned before the first row.

  – RELATIVE count

    Fetches the *count*'th succeeding row or the *count*'th prior row if count is negative.

    Value range: a possibly-signed integer

    ■ If *count* is positive, the *count*th succeeding row will be fetched.

    ■ If **count** is a negative integer, fetches the abs(count)'th prior row.

    ■ If the current row contains no data, **RELATIVE 0** returns null.

  – count

    Fetches the next *count* rows (same as **FORWARD** *count*).

  – ALL

    Fetches all remaining rows (same as **FORWARD ALL**).

  – FORWARD

    Fetches the next row (same as **NEXT**).

  – FORWARD count

    Fetches the *count* succeeding rows or *count* prior rows if *count* is negative.

- FORWARD ALL
  Fetches all remaining rows.
- BACKWARD
  Fetches the prior row (same as **PRIOR**).
- BACKWARD count
  Fetches the prior *count* rows (scanning backwards).
  Value range: a possibly-signed integer
  - If *count* is positive, the prior *count* rows will be fetched.
  - If *count* is a negative, the succeeding *abs (count)* rows will be fetched.
  - **BACKWARD 0** re-fetches the current row, if any.
- BACKWARD ALL
  Fetches all prior rows (scanning backwards).

- **{ FROM | IN } cursor_name**
  Specifies the cursor name using the keyword **FROM** or **IN**.
  Value range: an existing cursor name

Examples

```
-- (For the SELECT statement, traverse a table using a cursor.) Start a transaction.
openGauss=# START TRANSACTION;

-- Set up cursor1.
openGauss=# CURSOR cursor1 FOR SELECT * FROM tpcds.customer_address ORDER BY 1;

-- Fetch the first three rows in cursor1.
openGauss=# FETCH FORWARD 3 FROM cursor1;
 ca_address_sk |  ca_address_id   | ca_street_number |   ca_street_name   | ca_street_type  | ca_suite_number
|     ca_city     |    ca_county    | ca_state |   ca_zip   |  ca_country   | ca_gmt_offset |   ca_location_type
---------------+------------------+------------------+--------------------+-----------------+-----------------
+-----------------+-----------------+----------+------------+---------------+---------------+----------------------
             1 | AAAAAAAABAAAAAAA | 18               | Jackson            | Parkway         | Suite 280       |
Fairfield       | Maricopa County | AZ       | 86192      | United States |         -7.00 | condo
             2 | AAAAAAAACAAAAAAA | 362              | Washington 6th     | RD              | Suite 80        |
Fairview        | Taos County     | NM       | 85709      | United States |         -7.00 | condo
             3 | AAAAAAAADAAAAAAA | 585              | Dogwood Washington | Circle          | Suite Q         |
Pleasant Valley | York County     | PA       | 12477      | United States |         -5.00 | single family
(3 rows)

-- Close the cursor and commit the transaction.
openGauss=# CLOSE cursor1;

-- End the transaction.
openGauss=# END;

-- (For the VALUES clause, traverse the clause using a cursor.) Start a transaction.
openGauss=# START TRANSACTION;

-- Set up cursor2.
openGauss=# CURSOR cursor2 FOR VALUES(1,2),(0,3) ORDER BY 1;

-- Fetch the first two rows in cursor2.
openGauss=# FETCH FORWARD 2 FROM cursor2;
column1 | column2
---------+---------
0 |       3
1 |       2
```

```
(2 rows)

-- Close the cursor and commit the transaction.
openGauss=# CLOSE cursor2;

-- End the transaction.
openGauss=# END;

-- (WITH HOLD cursor) Start a transaction.
openGauss=# START TRANSACTION;

-- Set up a WITH HOLD cursor.
openGauss=# DECLARE cursor1 CURSOR WITH HOLD FOR SELECT * FROM tpcds.customer_address ORDER
BY 1;

-- Fetch the first two rows in cursor1.
openGauss=# FETCH FORWARD 2 FROM cursor1;
 ca_address_sk |  ca_address_id   | ca_street_number |   ca_street_name   | ca_street_type  | ca_suite_number
|     ca_city     |    ca_county     | ca_state |   ca_zip   |  ca_country   | ca_gmt_offset |   ca_location_type
---------------+------------------+------------------+--------------------+-----------------+-----------------
+-----------------+-----------------+----------+------------+---------------+---------------+----------------------
             1 | AAAAAAAABAAAAAAA | 18               | Jackson            | Parkway         | Suite 280       |
Fairfield       | Maricopa County | AZ       | 86192      | United States |         -7.00 | condo
             2 | AAAAAAAACAAAAAAA | 362              | Washington 6th     | RD              | Suite 80        |
Fairview        | Taos County     | NM       | 85709      | United States |         -7.00 | condo
(2 rows)

-- End the transaction.
openGauss=# END;

-- Fetch the next row in cursor1.
openGauss=# FETCH FORWARD 1 FROM cursor1;
 ca_address_sk |  ca_address_id   | ca_street_number |   ca_street_name   | ca_street_type  | ca_suite_number
|     ca_city     |    ca_county     | ca_state |   ca_zip   |  ca_country   | ca_gmt_offset |   ca_location_type
---------------+------------------+------------------+--------------------+-----------------+-----------------
+-----------------+-----------------+----------+------------+---------------+---------------+----------------------
             3 | AAAAAAAADAAAAAAA | 585              | Dogwood Washington | Circle          | Suite Q         |
Pleasant Valley | York County     | PA       | 12477      | United States |         -5.00 | single family
(1 row)

-- Close the cursor.
openGauss=# CLOSE cursor1;
```

Helpful Links

CLOSE and **MOVE**

11.14.150 GRANT

Function

GRANT grants permissions to roles and users.

GRANT is used in the following scenarios:

- **Granting system permissions to roles or users**

  System permissions are also called user attributes, including SYSADMIN, CREATEDB, CREATEROLE, AUDITADMIN, MONADMIN, OPRADMIN, POLADMIN, INHERIT, REPLICATION, VCADMIN, and LOGIN.

  They can be specified only by the CREATE ROLE or ALTER ROLE statement. The SYSADMIN permissions can be granted and revoked using **GRANT ALL PRIVILEGE** and **REVOKE ALL PRIVILEGE**, respectively. System permissions cannot be inherited by a user from a role, and cannot be granted using PUBLIC.

- **Granting database object permissions to roles or users**

  Grant permissions on a database object (table, view, column, database, function, schema, or tablespace) to a role or user.

  GRANT gives specific permissions on a database object to one or more roles. These permissions are added to those already granted, if any.

  The keyword PUBLIC indicates that the permissions are to be granted to all roles, including those that might be created later. **PUBLIC** can be thought of as an implicitly defined group that always includes all roles. Any particular role will have the sum of permissions granted directly to it, permissions granted to any role it is presently a member of, and permissions granted to **PUBLIC**.

  If **WITH GRANT OPTION** is specified, the recipient of the permission can in turn grant it to others. Without a grant option, the recipient cannot do that. This option cannot be granted to **PUBLIC**, which is a unique GaussDB attribute.

  GaussDB grants the permissions on objects of certain types to PUBLIC users. By default, permissions on tables, columns, sequences, foreign data sources, foreign servers, schemas, and tablespaces are not granted to PUBLIC users, but the following permissions are granted to PUBLIC users: CONNECT and CREATE TEMP TABLE permissions on databases, EXECUTE permission on functions, and USAGE permission on languages and data types (including domains). An object owner can revoke the default permissions granted to PUBLIC users and grant permissions to other users as needed. For security purposes, you are advised to create an object and set its permissions in the same transaction so that other users do not have time windows to use the object. In addition, you can restrict the permissions of the PUBLIC user group by referring to t"Permission Management" in *Security Hardening Guide*. These default permissions can be modified using the **ALTER DEFAULT PRIVILEGES** command.

  By default, an object owner has all permissions on the object. For security purposes, the owner can discard some permissions. However, the ALTER, DROP, COMMENT, INDEX, VACUUM, and re-grantable permissions of the object are inherent permissions implicitly owned by the owner.

- **Granting the permissions of one role or user to others**

  Grant the permissions of one role or user to others. In this case, every role or user can be regarded as a set of one or more database permissions.

  If **WITH ADMIN OPTION** is specified, the recipients can in turn grant the permissions to other roles or users or revoke the permissions they have granted to other roles or users. If recipients' permissions are changed or revoked later, the grantees' permissions will also change.

  Database administrators can grant or revoke permissions to or from any roles or users. Roles with the CREATEROLE permission can grant or revoke permissions to or from non-admin roles.

- **Granting ANY permissions to roles or users**

  Grant ANY permissions to a specified role or user. For details about the value range of the ANY permissions, see the syntax. If **WITH ADMIN OPTION** is specified, the grantee can grant the ANY permissions to or revoke them from other roles or users. The ANY permissions can be inherited by a role but cannot be granted to PUBLIC users. An initial user and the system administrator when separation of duties is disabled can grant the ANY permissions to or revoke them from any role or user.

Currently, the following ANY permissions are supported: CREATE ANY TABLE, ALTER ANY TABLE, DROP ANY TABLE, SELECT ANY TABLE, INSERT ANY TABLE, UPDATE ANY TABLE, DELETE ANY TABLE, CREATE ANY SEQUENCE, CREATE ANY INDEX, CREATE ANY FUNCTION, EXECUTE ANY FUNCTION, CREATE ANY PACKAGE, EXECUTE ANY PACKAGE, and CREATE ANY TYPE. For details about the ANY permission scope, see **Table 11-122**.

Precautions

- It is not allowed to grant the ANY permissions to PUBLIC users or revoke the ANY permissions from PUBLIC users.
- The ANY permissions are database permissions and are valid only for database objects that are granted with the permissions. For example, **SELECT ANY TABLE** only allows a user to view all user table data in the current database, but the user does not have the permission to view user tables in other databases.
- Even if a user is granted with the ANY permissions, the user cannot perform INSERT, DELETE, UPDATE, and SELECT operations on the objects of private users.
- The ANY permissions and the original permissions do not affect each other.
- If a user is granted with the CREATE ANY TABLE permission, the owner of a table created in a schema with the same name as the user is the creator of the schema. When the user performs other operations on the table, the user needs to be granted with the corresponding operation permission.
- Exercise caution when granting the CREATE ANY FUNMCTION permission to users to prevent other users from using SECURITY DEFINER functions for privilege escalation.

Syntax

- Grant the table or view access permission to a user or role.
```
GRANT { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | ALTER | DROP |
COMMENT | INDEX | VACUUM } [, ...]
     | ALL [ PRIVILEGES ] }
   ON { [ TABLE ] table_name [, ...]
      | ALL TABLES IN SCHEMA schema_name [, ...] }
   TO { [ GROUP ] role_name | PUBLIC } [, ...]
   [ WITH GRANT OPTION ];
```
- Grant the column access permission to a user or role.
```
GRANT { {{ SELECT | INSERT | UPDATE | REFERENCES | COMMENT } ( column_name [, ...] )} [, ...]
     | ALL [ PRIVILEGES ] ( column_name [, ...] ) }
   ON [ TABLE ] table_name [, ...]
   TO { [ GROUP ] role_name | PUBLIC } [, ...]
   [ WITH GRANT OPTION ];
```
- Grant the sequence access permission to a specified role or user. The **LARGE** field is optional. The assignment statement does not distinguish whether the sequence is LARGE.
```
GRANT { { SELECT | UPDATE | USAGE | ALTER | DROP | COMMENT } [, ...]
     | ALL [ PRIVILEGES ] }
   ON { [ [ LARGE ] SEQUENCE ] sequence_name [, ...]
      | ALL SEQUENCES IN SCHEMA schema_name [, ...] }
```

```
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

- Grant the database access permission to a user or role.

```
GRANT { { CREATE | CONNECT | TEMPORARY | TEMP | ALTER | DROP | COMMENT } [, ...]
      | ALL [ PRIVILEGES ] }
    ON DATABASE database_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

- Grant the domain access permission to a user or role.

```
GRANT { USAGE | ALL [ PRIVILEGES ] }
    ON DOMAIN domain_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

NOTE

In the current version, the domain access permission cannot be granted.

- Grant the client master key (CMK) access permission to a specified user or role.

```
GRANT { { USAGE | DROP } [, ...] | ALL [ PRIVILEGES ] }
    ON CLIENT_MASTER_KEY client_master_key [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

- Grant the column encryption key (CEK) access permission to a specified user or role.

```
GRANT { { USAGE | DROP } [, ...] | ALL [ PRIVILEGES ] }
    ON COLUMN_ENCRYPTION_KEY column_encryption_key [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

- Grant the foreign data source access permission to a user or role.

```
GRANT { USAGE | ALL [ PRIVILEGES ] }
    ON FOREIGN DATA WRAPPER fdw_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

- Grant the foreign server access permission to a user or role.

```
GRANT { { USAGE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON FOREIGN SERVER server_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

- Grant the function access permission to a user or role.

```
GRANT { { EXECUTE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON { FUNCTION {function_name ( [ {[ argmode ] [ arg_name ] arg_type} [, ...] ] )} [, ...]
       | ALL FUNCTIONS IN SCHEMA schema_name [, ...] }
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

- Grant the procedural procedure access permission to a user or role.

```
GRANT { { EXECUTE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON { PROCEDURE {proc_name ( [ {[ argmode ] [ arg_name ] arg_type} [, ...] ] )} [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

- Grant the procedural language access permission to a user or role.

```
GRANT { USAGE | ALL [ PRIVILEGES ] }
    ON LANGUAGE lang_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

- Grant the large object access permission to a specified user or role.

```
GRANT { { SELECT | UPDATE } [, ...] | ALL [ PRIVILEGES ] }
    ON LARGE OBJECT loid [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

NOTE

In the current version, the large object access permission cannot be granted.

- Grant the schema access permission to a user or role.

```
GRANT { { CREATE | USAGE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON SCHEMA schema_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

NOTE

When you grant table or view permissions to other users, you also need to grant the USAGE permission on the schema that the tables and views belong to. Without the USAGE permission, the users with table or view permissions can only see the object names, but cannot access them. This syntax cannot be used to grant the permission to create tables in schemas with the same name, but you can use the syntax for granting permission of a role to another user or role to achieve the same effect.

- Grant the tablespace access permission to a user or role.

```
GRANT { { CREATE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON TABLESPACE tablespace_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

- Grant the type access permission to a user or role.

```
GRANT { { USAGE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON TYPE type_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

NOTE

In the current version, the type access permission cannot be granted.

- Grant the data source permission to a role.

```
GRANT { USAGE | ALL [PRIVILEGES]}
   ON DATA SOURCE src_name [, ...]
   TO { [GROUP] role_name | PUBLIC } [, ...]
   [WITH GRANT OPTION];
```

- Grant the directory permission to a role.

```
GRANT { { READ | WRITE | ALTER | DROP } [, ...] | ALL [PRIVILEGES] }
   ON DIRECTORY directory_name [, ...]
   TO { [GROUP] role_name | PUBLIC } [, ...]
   [WITH GRANT OPTION];
```

- Grant the package permission to a role.

```
GRANT { { EXECUTE | ALTER | DROP | COMMENT } [, ...] | ALL [PRIVILEGES] }
   ON { PACKAGE package_name [, ...]
      | ALL PACKAGES IN SCHEMA schema_name [, ...] }
   TO { [GROUP] role_name | PUBLIC } [, ...]
   [WITH GRANT OPTION];
```

- Grant a role's permissions to another user or role.

```
GRANT role_name [, ...]
   TO role_name [, ...]
   [ WITH ADMIN OPTION ];
```

- Grant the sysadmin permission to a role.

```
GRANT ALL { PRIVILEGES | PRIVILEGE }
   TO role_name;
```

- Grant the ANY permissions to another user or role.

```
GRANT { CREATE ANY TABLE | ALTER ANY TABLE | DROP ANY TABLE | SELECT ANY TABLE | INSERT
ANY TABLE | UPDATE ANY TABLE |
  DELETE ANY TABLE | CREATE ANY SEQUENCE | CREATE ANY INDEX | CREATE ANY FUNCTION |
EXECUTE ANY FUNCTION |
  CREATE ANY PACKAGE | EXECUTE ANY PACKAGE | CREATE ANY TYPE } [, ...]
  TO [ GROUP ] role_name [, ...]
  [ WITH ADMIN OPTION ];
```

Parameter Description

The possible permissions are:

- **SELECT**

  Allows SELECT from any column, or the specific columns listed, of the specified table, view, or sequence. The SELECT permission on the corresponding field is also required for UPDATE or DELETE.
- **INSERT**

  Allows INSERT of a new row into a table.
- **UPDATE**

  Allows UPDATE of any column of a table. Generally, UPDATE also requires the SELECT permission to query which rows need to be updated. **SELECT ... FOR UPDATE** and **SELECT ... FOR SHARE** also require this permission on at least one column, in addition to the SELECT permission.
- **DELETE**

  Allows DELETE of a row from a table. Generally, DELETE also requires the SELECT permission to query which rows need to be deleted.
- **TRUNCATE**

  Allows TRUNCATE on a table.
- **REFERENCES**

  Allows creation of a foreign key constraint referencing a table. This permission is required on both referencing and referenced tables.
- **CREATE**
  - For databases, allows new schemas to be created within the database.
  - For schemas, allows new objects to be created within the schema. To rename an existing object, you must own the object and have the CREATE permission on the schema of the object.
  - For tablespaces, allows tables to be created within the tablespace, and allows databases and schemas to be created that have the tablespace as their default tablespace.
- **CONNECT**

  Allows the grantee to connect to the database.
- **EXECUTE**

  Allows calling a function, including use of any operators that are implemented on top of the function.
- **USAGE**
  - For procedural languages, allows use of the language for the creation of functions in that language.
  - For schemas, allows access to objects contained in the schema. Without this permission, it is still possible to see the object names.
  - For sequences, allows use of the **nextval** function.
  - For data sources, specifies access permissions or is used as **ALL PRIVILEGES**.
- **ALTER**

Allows users to modify the attributes of a specified object, excluding the owner and schema of the object.

- **DROP**

  Allows users to delete specified objects.
- **COMMENT**

  Allows users to define or modify comments of a specified object.
- **INDEX**

  Allows users to create indexes on specified tables, manage indexes on the tables, and perform REINDEX and CLUSTER operations on the tables.
- **VACUUM**

  Allows users to perform ANALYZE and VACUUM operations on specified tables.
- **ALL PRIVILEGES**

  Grants all available permissions to a user or role at a time. Only a system administrator has the GRANT ALL PRIVILEGES permission.

GRANT parameters are as follows:

- **role_name**

  Specifies the username.
- **table_name**

  Specifies the table name.
- **column_name**

  Specifies the column name.
- **schema_name**

  Specifies the schema name.
- **database_name**

  Specifies the database name.
- **function_name**

  Specifies the function name.
- **procedure_name**

  Specifies the stored procedure name.
- **sequence_name**

  Specifies the sequence name.
- **domain_name**

  Specifies the domain type name.
- **fdw_name**

  Specifies the foreign data wrapper name.
- **lang_name**

  Specifies the language name.
- **type_name**

  Specifies the type name.
- **src_name**

Specifies the data source name.

- **argmode**

  Specifies the parameter mode.

  Value range: a string. It must comply with the identifier naming convention.
- **arg_name**

  Specifies the parameter name.

  Value range: a string. It must comply with the identifier naming convention.
- **arg_type**

  Specifies the parameter type.

  Value range: a string. It must comply with the identifier naming convention.
- **loid**

  Specifies the identifier of the large object that includes this page.

  Value range: a string. It must comply with the identifier naming convention.
- **tablespace_name**

  Specifies the tablespace name.
- client_master_key

  Name of the client master key.

  Value range: a string. It must comply with the identifier naming convention.
- column_encryption_key

  Name of the column encryption key.

  Value range: a string. It must comply with the identifier naming convention.
- **directory_name**

  Specifies the directory name.

  Value range: a string. It must comply with the identifier naming convention.
- **WITH GRANT OPTION**

  If **WITH GRANT OPTION** is specified, the recipient of the permission can in turn grant it to others. Without a grant option, the recipient cannot do that. Grant options cannot be granted to **PUBLIC**.

When a non-owner of an object attempts to GRANT permissions on the object:

- The statement will fail outright if the user has no permissions whatsoever on the object.
- As long as some permission is available, the statement will proceed, but it will grant only those permissions for which the user has grant options.
- The **GRANT ALL PRIVILEGES** forms will issue a warning message if no grant options are held, while the other forms will issue a warning if grant options for any of the permissions specifically named in the statement are not held.

NOTE

Database administrators can access all objects, regardless of object permission settings. This is comparable to the permissions of **root** in a Unix system. As with **root**, it is unwise to operate as a system administrator except when necessary.

- WITH ADMIN OPTION

If **WITH ADMIN OPTION** is specified for a role, the grantee can grant the role to other roles or users or revoke the role from other roles or users.

For the ANY permissions, if **WITH ADMIN OPTION** is specified, the grantee can grant the ANY permissions to or revoke them from other roles or users.

Table 11-122 ANY permissions

System Permission	Description
CREATE ANY TABLE	Users can create tables or views in the public and user schemas. The users must be granted with the permission to create sequences to create a table that contains serial columns.
ALTER ANY TABLE	Users' **ALTER** permission on tables or views in the public and user schemas. If the users want to modify the unique index of a table to add a primary key constraint or unique constraint to the table, the users must be granted with the index permission for the table.
DROP ANY TABLE	Users' DROP permission on tables or views in the public and user schemas.
SELECT ANY TABLE	Users' SELECT permission on tables or views in the public and user schemas, which is still subject to row-level access control.
UPDATE ANY TABLE	Users' UPDATE permission on tables or views in the public and user schemas, which is still subject to row-level access control.
INSERT ANY TABLE	Users' INSERT permission on tables or views in the public and user schemas.
DELETE ANY TABLE	Users' DELETE permission on tables or views in the public and user schemas, which is still subject to row-level access control.
CREATE ANY FUNCTION	Users can create functions or stored procedures in the user schemas.
EXECUTE ANY FUNCTION	Users' EXECUTE permission on functions or stored procedures in the public and user schemas.
CREATE ANY PACKAGE	Users can create packages in the public and user schemas.
EXECUTE ANY PACKAGE	Users' EXECUTE permission on packages in the public and user schemas.
CREATE ANY TYPE	Users can create types in the public and user schemas.
CREATE ANY SEQUENCE	Users can create sequences in the public and user schemas.

System Permission	Description
CREATE ANY INDEX	Users can create indexes in the public and user schemas. The users must be granted with the permission to create tablespaces to create a partitioned table index in a tablespace.

NOTE

If a user is granted with any ANY permission, the user has the USAGE permission on the public and user schemas but does not have the USAGE permission on the system schemas except **public** listed in **Table 14-1**.

Examples

Example: Granting system permissions to a user or role

Create the **joe** user and grant the sysadmin permissions to it.

```
openGauss=# CREATE USER joe PASSWORD 'xxxxxxxxx';
openGauss=# GRANT ALL PRIVILEGES TO joe;
```

Then **joe** has the sysadmin permission.

Example: Granting object permissions to a user or role

1. Revoke the sysadmin permission from the **joe** user. Grant the usage permission of the **tpcds** schema and all permissions on the **tpcds.reason** table to **joe**.
   ```
   openGauss=# REVOKE ALL PRIVILEGES FROM joe;
   openGauss=# GRANT USAGE ON SCHEMA tpcds TO joe;
   openGauss=# GRANT ALL PRIVILEGES ON tpcds.reason TO joe;
   ```
   Then **joe** has all permissions on the **tpcds.reason** table, including create, retrieve, update, and delete.
2. Grant the retrieve permission of **r_reason_sk**, **r_reason_id**, and **r_reason_desc** columns and the update permission of the **r_reason_desc** column in the **tpcds.reason** table to **joe**.
   ```
   openGauss=# GRANT select (r_reason_sk,r_reason_id,r_reason_desc),update (r_reason_desc) ON
   tpcds.reason TO joe;
   ```
   Then **joe** has the retrieve permission of **r_reason_sk** and **r_reason_id** columns in the **tpcds.reason** table. To enable **joe** to grant these permissions to other users, execute the following statement:
   ```
   openGauss=# GRANT select (r_reason_sk, r_reason_id) ON tpcds.reason TO joe WITH GRANT OPTION;
   ```
   Grant the database connection permission and the permission to create schemas in GaussDB to user **joe**, and allow user **joe** to grant this permission to other users.
   ```
   openGauss=# GRANT create,connect on database openGauss TO joe WITH GRANT OPTION;
   ```
   Create the **tpcds_manager** role, grant the access and object creation permissions of the **tpcds** schema to **tpcds_manager**, but do not allow **tpcds_manager** to grant these permissions to others.
   ```
   openGauss=# CREATE ROLE tpcds_manager PASSWORD 'xxxxxxxxx';
   openGauss=# GRANT USAGE,CREATE ON SCHEMA tpcds TO tpcds_manager;
   ```

Grant all permissions on the **tpcds_tbspc** tablespace to **joe**, but do not allow **joe** to grant these permissions to others.

```
openGauss=# CREATE TABLESPACE tpcds_tbspc RELATIVE LOCATION 'tablespace/tablespace_1';
openGauss=# GRANT ALL ON TABLESPACE tpcds_tbspc TO joe;
```

Example: Granting the permissions of one user or role to others

1. Create the **manager** role, grant **joe**'s permissions to **manager**, and allow **manager** to grant these permissions to others.
```
openGauss=# CREATE ROLE manager PASSWORD 'xxxxxxxxx';
openGauss=# GRANT joe TO manager WITH ADMIN OPTION;
```
2. Create the **senior_manager** user and grant **manager**'s permissions to it.
```
openGauss=# CREATE ROLE senior_manager PASSWORD 'xxxxxxxxx';
openGauss=# GRANT manager TO senior_manager;
```
3. Revoke permissions and delete users.
```
openGauss=# REVOKE manager FROM joe;
openGauss=# REVOKE senior_manager FROM manager;
openGauss=# DROP USER manager;
```

Example: Granting the CMK or CEK permission to other user or role

1. Connect to an encrypted database.
```
gsql -p 57101 openGauss -r -C
openGauss=#  CREATE CLIENT MASTER KEY MyCMK1 WITH ( KEY_STORE = gs_ktool, KEY_PATH =
"gs_ktool/1" , ALGORITHM = AES_256_CBC);
CREATE CLIENT MASTER KEY
openGauss=# CREATE COLUMN ENCRYPTION KEY MyCEK1 WITH VALUES (CLIENT_MASTER_KEY =
MyCMK1, ALGORITHM = AEAD_AES_256_CBC_HMAC_SHA256);
CREATE COLUMN ENCRYPTION KEY
```
2. Create a role **newuser** and grant the key permission to **newuser**.
```
openGauss=# CREATE USER newuser PASSWORD 'xxxxxxxxx';
CREATE ROLE
openGauss=# GRANT ALL ON SCHEMA public TO newuser;
GRANT
openGauss=# GRANT USAGE ON COLUMN_ENCRYPTION_KEY MyCEK1 to newuser;
GRANT
openGauss=# GRANT USAGE ON CLIENT_MASTER_KEY MyCMK1 to newuser;
GRANT
```
3. Set the user to connect to a database and use a CEK to create an encrypted table.
```
openGauss=# SET SESSION AUTHORIZATION newuser PASSWORD 'xxxxxxxxx';
openGauss=>  CREATE TABLE acltest1 (x int, x2 varchar(50) ENCRYPTED WITH
(COLUMN_ENCRYPTION_KEY = MyCEK1, ENCRYPTION_TYPE = DETERMINISTIC));
CREATE TABLE
openGauss=> SELECT has_cek_privilege('newuser', 'MyCEK1', 'USAGE');
 has_cek_privilege
-------------------
 t
(1 row)
```
4. Revoke permissions and delete users.
```
openGauss=# REVOKE USAGE ON COLUMN_ENCRYPTION_KEY MyCEK1 FROM newuser;
openGauss=# REVOKE USAGE ON CLIENT_MASTER_KEY MyCMK1 FROM newuser;
openGauss=# DROP TABLE newuser.acltest1;
openGauss=# DROP COLUMN ENCRYPTION KEY MyCEK1;
openGauss=# DROP CLIENT MASTER KEY MyCMK1;
openGauss=# DROP SCHEMA IF EXISTS newuser CASCADE;
openGauss=# REVOKE ALL ON SCHEMA public FROM newuser;
openGauss=# DROP ROLE IF EXISTS newuser;
```

Example: Revoking permissions and deleting roles and users

```
openGauss=# REVOKE ALL PRIVILEGES ON tpcds.reason FROM joe;
openGauss=# REVOKE ALL PRIVILEGES ON SCHEMA tpcds FROM joe;
openGauss=# REVOKE ALL ON TABLESPACE tpcds_tbspc FROM joe;
```

```
openGauss=# DROP TABLESPACE tpcds_tbspc;
openGauss=# REVOKE USAGE,CREATE ON SCHEMA tpcds FROM tpcds_manager;
openGauss=# DROP ROLE tpcds_manager;
openGauss=# DROP ROLE senior_manager;
openGauss=# DROP USER joe CASCADE;
```

Helpful Links

REVOKE and **ALTER DEFAULT PRIVILEGES**.

11.14.151 INSERT

Function

INSERT inserts new rows into a table.

Precautions

- You must have the **INSERT** permission on a table to insert data to it. If a user is granted with the **INSERY ANY TABLE** permission, the user has the **USAGE** permission on all schemas except system schemas and the **INSERT** permission on tables in these schemas.
- Use of the **RETURNING** clause requires the **SELECT** permission on all columns mentioned in **RETURNING**.
- For column-store tables, the **RETURNING** clause is currently not supported.
- If **ON DUPLICATE KEY UPDATE** is used, you must have the **SELECT** and **UPDATE** permissions on the table and the **SELECT** permission on the unique constraint (primary key or unique index).
- If you use the **query** clause to insert rows from a query, you need to have the **SELECT** permission on any table or column used in the query.
- The generated column cannot be directly written. In the INSERT statement, values cannot be specified for generated columns, but the keyword DEFAULT can be specified.
- When you connect to a database compatible to Teradata and **td_compatible_truncation** is **on**, a long string will be automatically truncated. If later **INSERT** statements (not involving foreign tables) insert long strings to columns of char- and varchar-typed columns in the target table, the system will truncate the long strings to ensure no strings exceed the maximum length defined in the target table.

  NOTE

  If inserting multi-byte character data (such as Chinese characters) to a database with the character set byte encoding (SQL_ASCII, LATIN1), and the character data crosses the truncation position, the string is truncated based on its bytes instead of characters. Unexpected result will occur in tail after the truncation. If you want correct truncation result, you are advised to adopt encoding set such as UTF8, which has no character data crossing the truncation position.

Syntax

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
INSERT [/*+ plan_hint */] INTO table_name [partition_clause] [ AS alias ] [ ( column_name [, ...] ) ]
    { DEFAULT VALUES
    | VALUES {( { expression | DEFAULT } [, ...] ) }[, ...]
    | query }
    [ ON DUPLICATE KEY UPDATE { NOTHING | { column_name = { expression | DEFAULT } } [, ...] [ WHERE
condition ] }]
    [ RETURNING {* | {output_expression [ [ AS ] output_name ] }[, ...]} ];
```

Parameter Description

- **WITH [RECURSIVE] with_query [, ...]**

  Specifies one or more subqueries that can be referenced by name in the main query, which is equivalent to a temporary table.

  If **RECURSIVE** is specified, it allows a **SELECT** subquery to reference itself by name.

  Format of **with_query**:

  ```
  with_query_name [ ( column_name [, ...] ) ] AS [ [ NOT ] MATERIALIZED ]
  ( {select | values | insert | update | delete} )
  ```

  – **with_query_name** specifies the name of the result set generated by a subquery. Such names can be used to access the result sets of subqueries in a query.

  -- **column_name** specifies the column name displayed in the subquery result set.

  Each subquery can be a **SELECT**, **VALUES**, **INSERT**, **UPDATE** or **DELETE** statement.

  – You can use **MATERIALIZED** or **NOT MATERIALIZED** to modify the CTE.

  – If **MATERIALIZED** is specified, the WITH query will be materialized, and a copy of the subquery result set is generated. The copy is directly queried at the reference point. Therefore, the WITH subquery cannot be jointly optimized with the SELECT statement trunk (for example, predicate pushdown and equivalence class transfer). In this scenario, you can use **NOT MATERIALIZED** for modification. If the WITH query can be executed as a subquery inline, the preceding optimization can be performed.

  – If the user does not explicitly declare the materialized attribute, comply with the following rules: If the CTE is referenced only once in the trunk statement to which it belongs and semantically supports inline execution, it will be rewritten as subquery inline execution. Otherwise, the materialized execution will be performed in CTE Scan mode.

  NOTE

  INSERT ON DUPLICATE KEY UPDATE does not support the **WITH** and **WITH RECURSIVE** clauses.

- **plan_hint** clause

  Follows the **INSERT** keyword in the **/*+ */** format. It is used to optimize the plan of an **INSERT** statement block. For details, see **Hint-based Tuning**. In each statement, only the first **/*+** *plan_hint* ***/** comment block takes effect as a hint. Multiple hints can be written.

- **table_name**

  Specifies the name of the target table where data will be inserted.

  Value range: an existing table name

- **partition_clause**

  Inserts data to a specified partition.

```
PARTITION { ( partition_name ) | FOR ( partition_value [, ...] ) } |
SUBPARTITION { ( subpartition_name ) | FOR ( subpartition_value [, ...] ) }
```

For details about the keywords, see **SELECT**.

If the value of the **value** clause is inconsistent with the specified partition, an error is reported.

For details, see **CREATE TABLE SUBPARTITION**.

- **column_name**

  Specifies the name of a column in a table.

  – The column name can be qualified with a subfield name or array subscript, if needed.

  – Each column not present in the explicit or implicit column list will be filled with a default value, either its declared default value or **NULL** if there is none. Inserting into only some fields of a composite column leaves the other fields null.

  – The target column names **column_name** can be listed in any order. If no list of column names is given at all, the default is all the columns of the table in their declared order.

  – The target columns are the first *N* column names, if there are only *N* columns supplied by the **value** clause or **query**.

  – The values provided by the **value** clause and query are associated with the corresponding columns from left to right in the table.

  Value range: an existing column

- **expression**

  Specifies an expression or a value to assign to the corresponding column.

  – In the **INSERT ON DUPLICATE KEY UPDATE** statement, expression can be **VALUES(***column_name***)** or **EXCLUDED.***column_name*, indicating that the value of **column_name** corresponding to the conflict row is referenced. Note that **VALUES(***column_name***)** cannot be nested in an expression (for example, **VALUES(***column_name***)+1**). **EXCLUDED** is not subject to this restriction.

  – If single-quotation marks are inserted in a column, the single-quotation marks need to be used for escape.

  – If the expression for any column is not of the correct data type, automatic type conversion will be attempted. If the attempt fails, data insertion fails, and the system returns an error message.

- **DEFAULT**

  Specifies the default value of a field. The value is **NULL** if no default value is assigned to it.

- **query**

  Specifies a query statement (SELECT statement) that uses the query result as the inserted data.

- **RETURNING**

  Returns the inserted rows. The syntax of the **RETURNING** list is identical to that of the output list of **SELECT**. Note that **INSERT ON DUPLICATE KEY UPDATE** does not support the **RETURNING** clause.

- **output_expression**

Specifies an expression used to calculate the output result of the **INSERT** statement after each row is inserted.

Value range: The expression can use any field in the table. You can use the asterisk (*) to return all fields of the inserted row.

- **output_name**

  Specifies a name to use for a returned column.

  Value range: a string. It must comply with the identifier naming convention.

- **ON DUPLICATE KEY UPDATE**

  For a table with a unique constraint (**UNIQUE INDEX** or **PRIMARY KEY**), if the inserted data violates the unique constraint, the **UPDATE** clause is executed on the conflicting row to complete the update. For a table without a unique constraint, only the insert operation is performed. When **UPDATE** is used, if **NOTHING** is specified, this insertion is ignored. You can use **EXCLUDE.** or **VALUES()** to select the column corresponding to the source data.

  - Triggers are supported. The execution sequence of triggers is determined by the actual execution process.
    - Run the **insert** command to trigger the **before insert** and **after insert** triggers.
    - Run the **update** command to trigger the **before insert**, **before update**, and **after update** triggers.
    - Run the **update nothing** command to trigger the **before insert** trigger.
  - The unique constraint or primary key of **DEFERRABLE** is not supported.
  - If a table has multiple unique constraints and the inserted data violates multiple unique constraints, only the first row that has a conflict is updated. (The check sequence is closely related to index maintenance. Generally, the conflict check is performed on the index that is created first.)
  - If multiple rows are inserted and these rows conflict with the same row in the table, the system inserts or updates the first row and then updates other rows in sequence.
  - Primary keys and unique index columns cannot be updated.
  - Column-store tables, foreign tables, and memory tables are not supported.
  - Subquery expressions are supported. The syntax and function of the expressions are the same as those of **UPDATE**. In a subquery expression, **EXCLUDED** can be used to select the columns corresponding to the source data.

Examples

```
-- Create the tpcds.reason_t2 table.
openGauss=# CREATE TABLE tpcds.reason_t2
(
  r_reason_sk    integer,
  r_reason_id    character(16),
  r_reason_desc  character(100)
);
```

```
-- Insert a record into a table:
openGauss=# INSERT INTO tpcds.reason_t2(r_reason_sk, r_reason_id, r_reason_desc) VALUES (1, 
'AAAAAAAABAAAAAAA', 'reason1');

-- Insert a record into the table, which is equivalent to the previous syntax:
openGauss=# INSERT INTO tpcds.reason_t2 VALUES (2, 'AAAAAAAABAAAAAAA', 'reason2');

-- Insert multiple records into the table.
openGauss=# INSERT INTO tpcds.reason_t2 VALUES (3, 'AAAAAAAACAAAAAAA','reason3'),(4, 
'AAAAAAAADAAAAAAA', 'reason4'),(5, 'AAAAAAAAEAAAAAAA','reason5');

-- Insert records whose r_reason_sk in the tpcds.reason table is less than 5.
openGauss=# INSERT INTO tpcds.reason_t2 SELECT * FROM tpcds.reason WHERE r_reason_sk <5;

-- Create a unique index for the table:
openGauss=# CREATE UNIQUE INDEX reason_t2_u_index ON tpcds.reason_t2(r_reason_sk);

-- Insert multiple records into the table. If the records conflict, update the r_reason_id field in the conflict 
data row to BBBBBBBBCAAAAAAA.
openGauss=# INSERT INTO tpcds.reason_t2 VALUES (5, 'BBBBBBBBCAAAAAAA','reason5'),(6, 
'AAAAAAAADAAAAAAA', 'reason6') ON DUPLICATE KEY UPDATE r_reason_id = 'BBBBBBBBCAAAAAAA';

-- Delete the tpcds.reason_t2.
openGauss=# DROP TABLE tpcds.reason_t2;
```

Suggestions

- VALUES

  When you run the **INSERT** statement to insert data in batches, you are advised to combine multiple records into one statement to improve data loading performance. Example: **INSERT INTO sections VALUES (30, 'Administration', 31, 1900),(40, 'Development', 35, 2000), (50, 'Development' , 60 , 2001);**

11.14.152 LOCK

Function

LOCK TABLE obtains a table-level lock.

GaussDB always tries to select the lock mode with minimum constraints when automatically requesting a lock for a statement referenced by a table. Use **LOCK** if users need a more strict lock mode. For example, suppose an application runs a transaction at the Read Committed isolation level and needs to ensure that data in a table remains stable in the duration of the transaction. To achieve this, you could obtain **SHARE** lock mode over the table before the query. This will prevent concurrent data changes and ensure subsequent reads of the table see a stable view of committed data. It is because the **SHARE** lock mode conflicts with the **ROW EXCLUSIVE** lock acquired by writers, and your **LOCK TABLE name IN SHARE MODE** statement will wait until any concurrent holders of **ROW EXCLUSIVE** mode locks commit or roll back. Therefore, once you obtain the lock, there are no uncommitted writes outstanding; furthermore none can begin until you release the lock.

Precautions

- **LOCK TABLE** is useless outside a transaction block: the lock would remain held only to the completion of the statement. If **LOCK TABLE** is out of any transaction block, an error is reported.

- If no lock mode is specified, then **ACCESS EXCLUSIVE**, the most restrictive mode, is used.
- LOCK TABLE ... **IN ACCESS SHARE MODE** requires the **SELECT** permission on the target table. All other forms of **LOCK** require table-level **UPDATE** and/or the **DELETE** permission.
- There is no **UNLOCK TABLE** statement. Locks are always released at transaction end.
- **LOCK TABLE** only deals with table-level locks, and so the mode names involving **ROW** are all misnomers. These mode names should generally be read as indicating the intention of the user to acquire row-level locks within the locked table. Also, **ROW EXCLUSIVE** mode is a shareable table lock. Note that all the lock modes have identical semantics so far as **LOCK TABLE** is concerned, differing only in the rules about which modes conflict with which. For details about the rules, see **Table 11-123**.
- If the xc_maintenance_mode parameter is not enabled, an error is reported when an ACCESS EXCLUSIVE lock is applied for a system catalog.

Syntax

```
LOCK [ TABLE ] {[ ONLY ] name [, ...]| {name [ * ]} [, ...]}
    [ IN {ACCESS SHARE | ROW SHARE | ROW EXCLUSIVE | SHARE UPDATE EXCLUSIVE | SHARE | SHARE
ROW EXCLUSIVE | EXCLUSIVE | ACCESS EXCLUSIVE} MODE ]
    [ NOWAIT ];
```

Parameter Description

Table 11-123 Lock mode conflicts

Requested Lock Mode/Current Lock Mode	ACCESS SHARE	ROW SHARE	ROW EXCLUSIVE	SHARE UPDATE EXCLUSIVE	SHARE	SHARE ROW EXCLUSIVE	EXCLUSIVE	ACCESS EXCLUSIVE
ACCESS SHARE	-	-	-	-	-	-	-	X
ROW SHARE	-	-	-	-	-	-	X	X
ROW EXCLUSIVE	-	-	-	-	X	X	X	X
SHARE UPDATE EXCLUSIVE	-	-	-	X	X	X	X	X

Requested Lock Mode/Current Lock Mode	ACCESS SHARE	ROW SHARE	ROW EXCLUSIVE	SHARE UPDATE EXCLUSIVE	SHARE	SHARE ROW EXCLUSIVE	EXCLUSIVE	ACCESS EXCLUSIVE
SHARE	-	-	X	X	-	X	X	X
SHARE ROW EXCLUSIVE	-	-	X	X	X	X	X	X
EXCLUSIVE	-	X	X	X	X	X	X	X
ACCESS EXCLUSIVE	X	X	X	X	X	X	X	X

LOCK parameters are as follows:

- **name**

  Specifies the name (optionally schema-qualified) of an existing table to lock.

  Tables are locked one-by-one in the order specified in the **LOCK TABLE** statement.

  Value range: an existing table name
- **ONLY**

  If **ONLY** is specified, only that table is locked. If **ONLY** is not specified, the table and all its sub-tables are locked.
- **ACCESS SHARE**

  Conflicts with the ACCESS EXCLUSIVE lock mode only.

  The **SELECT** statement acquires a lock of this mode on referenced tables. Typically, any command that reads a table without modifying it acquires this lock mode.
- **ROW SHARE**

  It conflicts with the EXCLUSIVE and ACCESS EXCLUSIVE lock modes.

  SELECT FOR UPDATE and **SELECT FOR SHARE** automatically acquire the **ROW SHARE** lock on the target table and add the **ACCESS SHARE** lock to other referenced tables except **FOR SHARE** and **FOR UPDATE**.
- **ROW EXCLUSIVE**

  Allows concurrent read of a table but does not allow modification of data in the table like **ROW SHARE**. **UPDATE**, **DELETE**, and **INSERT** automatically acquire the **ROW SHARE** lock on the target table and add the **ACCESS**

SHARE lock to other referenced tables. Generally, all statements that modify table data acquire the **ROW EXCLUSIVE** lock for tables.

- **SHARE UPDATE EXCLUSIVE**

  Protects a table against concurrent schema changes and VACUUM runs.

  Acquired by **VACUUM** (without **FULL**), **ANALYZE** and **CREATE INDEX CONCURRENTLY** statements, and some forms of **ALTER TABLE**.

- **SHARE**

  Allows concurrent queries of a table but does not allow modification of the table.

  Acquired by **CREATE INDEX** (without **CONCURRENTLY**).

- **SHARE ROW EXCLUSIVE**

  Protects a table against concurrent data changes, and is self-exclusive so that only one session can hold it at a time.

  No SQL statements automatically acquire this lock mode.

- **EXCLUSIVE**

  Allows concurrent queries of the target table but does not allow any other operations.

  This mode allows only concurrent **ACCESS SHARE** locks; that is, only reads from the table can proceed in parallel with a transaction holding this lock mode.

  No SQL statements automatically acquire this lock mode on user tables. However, it will be acquired on some system catalogs in case of some operations.

- **ACCESS EXCLUSIVE**

  Guarantees that the holder is the only transaction accessing the table in any way.

  Acquired by the **ALTER TABLE**, **DROP TABLE**, **TRUNCATE**, and **REINDEX** statements.

  This is also the default lock mode for **LOCK TABLE** statements that do not specify a mode explicitly.

- **NOWAIT**

  Specifies that **LOCK TABLE** should not wait for any conflicting locks to be released: if the specified lock(s) cannot be acquired immediately without waiting, the transaction is aborted.

  If **NOWAIT** is not specified, **LOCK TABLE** obtains a table-level lock, waiting if necessary for any conflicting locks to be released.

Examples

```
-- Obtain a SHARE ROW EXCLUSIVE lock on a primary key table when going to perform a delete operation.
openGauss=# CREATE TABLE tpcds.reason_t1 AS TABLE tpcds.reason;

openGauss=# START TRANSACTION;

openGauss=# LOCK TABLE tpcds.reason_t1 IN SHARE ROW EXCLUSIVE MODE;

openGauss=# DELETE FROM tpcds.reason_t1 WHERE r_reason_desc IN(SELECT r_reason_desc FROM
tpcds.reason_t1 WHERE r_reason_sk < 6 );

openGauss=# DELETE FROM tpcds.reason_t1 WHERE r_reason_sk = 7;
```

```
openGauss=# COMMIT;

-- Delete the tpcds.reason_t1 table.
openGauss=# DROP TABLE tpcds.reason_t1;
```

11.14.153 MERGE INTO

Function

MERGE INTO conditionally matches data in a target table with that in a source table. If data matches, **UPDATE** is executed on the target table; if data does not match, **INSERT** is executed. You can use this syntax to run **UPDATE** and **INSERT** at a time for convenience

Precautions

You have the **INSERT** and **UPDATE** permissions for the target table and the **SELECT** permission for the source table.

Syntax

```
MERGE [/*+ plan_hint */] INTO table_name [ partition_clause ] [ [ AS ] alias ]
USING { { table_name | view_name } | subquery } [ [ AS ] alias ]
ON ( condition )
[
  WHEN MATCHED THEN
  UPDATE SET { column_name = { expression | subquery | DEFAULT } |
          ( column_name [, ...] ) = ( { expression | subquery | DEFAULT } [, ...] ) } [, ...]
  [ WHERE condition ]
]
[
  WHEN NOT MATCHED THEN
  INSERT { DEFAULT VALUES |
  [ ( column_name [, ...] ) ] VALUES ( { expression | subquery | DEFAULT } [, ...] ) [, ...] [ WHERE condition ] }
];
NOTICE: 'subquery' in the UPDATE and INSERT clauses are only avaliable in CENTRALIZED mode!
```

Parameter Description

- **plan_hint** clause

  Follows the **MERGE** keyword in the **/*+ */** format. It is used to optimize the plan of a **MERGE** statement block. For details, see **Hint-based Tuning**. In each statement, only the first **/*+** *plan_hint* ***/** comment block takes effect as a hint. Multiple hints can be written.

- **INTO** clause

  Specifies the target table that is being updated or has data being inserted.

- **talbe_name**

  Specifies the name of the target table.

- **partition_clause**

  Performs MERGE operations on a specified partition.

  ```
  PARTITION { ( partition_name ) | FOR ( partition_value [, ...] ) } |
  SUBPARTITION { ( subpartition_name ) | FOR ( subpartition_value [, ...] ) }
  ```

  For details about the keywords, see **SELECT**.

  If the value of the **value** clause is inconsistent with the specified partition, an error is reported.

For details, see **CREATE TABLE SUBPARTITION**.

- **alias**

  Specifies the alias of the target table.

  Value range: a string. It must comply with the identifier naming convention.

- **USING** clause

  Specifies the source table, which can be a table, view, or subquery.

- **ON** clause

  Specifies the condition used to match data between the source and target tables. Columns in the condition cannot be updated.

- **WHEN MATCHED** clause

  Performs **UPDATE** if data in the source table matches that in the target table based on the condition.

  System catalogs and system columns cannot be updated.

- **WHEN NOT MATCHED** clause

  Performs **INSERT** if data in the source table does not match that in the target table based on the condition.

  An **INSERT** clause can contain only one **VALUES**.

  The order of **WHEN MATCHED** and **WHEN NOT MATCHED** clauses can be reversed. One of them can be used by default, but they cannot be both used at one time. Two **WHEN MATCHED** or **WHEN NOT MATCHED** clauses cannot be specified at the same time.

- **DEFAULT**

  Specifies the default value of a column.

  The value is **NULL** if no default value is assigned to it.

- **WHERE condition**

  Specifies the conditions for the **UPDATE** and **INSERT** clauses. The two clauses will be executed only when the conditions are met. The default value can be used. System columns cannot be referenced in **WHERE condition**. You are not advised to use numeric types such as int for **condition**, because such types can be implicitly converted to bool values (non-zero values are implicitly converted to **true** and **0** is implicitly converted to **false**), which may cause unexpected results.

Examples

```
-- Create the target table products and source table newproducts, and insert data to them.
openGauss=# CREATE TABLE products
(
product_id INTEGER,
product_name VARCHAR2(60),
category VARCHAR2(60)
);

openGauss=# INSERT INTO products VALUES (1501, 'vivitar 35mm', 'electrncs');
openGauss=# INSERT INTO products VALUES (1502, 'olympus is50', 'electrncs');
openGauss=# INSERT INTO products VALUES (1600, 'play gym', 'toys');
openGauss=# INSERT INTO products VALUES (1601, 'lamaze', 'toys');
openGauss=# INSERT INTO products VALUES (1666, 'harry potter', 'dvd');

openGauss=# CREATE TABLE newproducts
(
product_id INTEGER,
product_name VARCHAR2(60),
category VARCHAR2(60)
);

openGauss=# INSERT INTO newproducts VALUES (1502, 'olympus camera', 'electrncs');
openGauss=# INSERT INTO newproducts VALUES (1601, 'lamaze', 'toys');
openGauss=# INSERT INTO newproducts VALUES (1666, 'harry potter', 'toys');
openGauss=# INSERT INTO newproducts VALUES (1700, 'wait interface', 'books');

-- Run MERGE INTO.
openGauss=# MERGE INTO products p
USING newproducts np
ON (p.product_id = np.product_id)
WHEN MATCHED THEN
  UPDATE SET p.product_name = np.product_name, p.category = np.category WHERE p.product_name !=
'play gym'
WHEN NOT MATCHED THEN
  INSERT VALUES (np.product_id, np.product_name, np.category) WHERE np.category = 'books';
MERGE 4

-- Query updates.
openGauss=# SELECT * FROM products ORDER BY product_id;
 product_id |  product_name  | category
------------+----------------+-----------
       1501 | vivitar 35mm   | electrncs
       1502 | olympus camera | electrncs
       1600 | play gym       | toys
       1601 | lamaze         | toys
       1666 | harry potter   | toys
       1700 | wait interface | books
(6 rows)

-- Delete the table.
openGauss=# DROP TABLE products;
openGauss=# DROP TABLE newproducts;
```

11.14.154 MOVE

Function

MOVE repositions a cursor without retrieving any data. **MOVE** works exactly like the **FETCH** statement, except it only repositions the cursor and does not return rows.

Precautions

None

Syntax

```
MOVE [ direction [ FROM | IN ] ] cursor_name;
```

The **direction** clause specifies optional parameters.

```
NEXT
   | PRIOR
   | FIRST
   | LAST
   | ABSOLUTE count
   | RELATIVE count
   | count
   | ALL
   | FORWARD
   | FORWARD count
```

```
  | FORWARD ALL
  | BACKWARD
  | BACKWARD count
  | BACKWARD ALL
```

Parameter Description

The parameters of **MOVE** and **FETCH** are the same. For details, see **Parameter Description** in **FETCH**.

NOTE

On successful completion, a **MOVE** statement returns a tag of the form **MOVE count**. The **count** is the number of rows that a **FETCH** statement with the same parameters would have returned (possibly zero).

Examples

```
-- Start a transaction.
openGauss=# START TRANSACTION;

-- Define a cursor cursor1.
openGauss=# CURSOR cursor1 FOR SELECT * FROM tpcds.reason;

-- Skip the first three rows of cursor1.
openGauss=# MOVE FORWARD 3 FROM cursor1;

-- Fetch the first four rows from cursor1.
openGauss=# FETCH 4 FROM cursor1;
 r_reason_sk |   r_reason_id    |                                    r_reason_desc
-------------+------------------
+----------------------------------------------------------------------------------------------------
           4 | AAAAAAAAEAAAAAAA | Not the product that was
ordred
           5 | AAAAAAAAFAAAAAAA | Parts missing
           6 | AAAAAAAAGAAAAAAA | Does not work with a product that I
have
           7 | AAAAAAAAHAAAAAAA | Gift
exchange
(4 rows)

-- Close the cursor.
openGauss=# CLOSE cursor1;

-- End the transaction.
openGauss=# END;
```

Helpful Links

CLOSE and **FETCH**

11.14.155 PREDICT BY

Function

PREDICT BY uses a trained model to perform inference tasks.

Precautions

The name of the invoked model can be viewed in the **gs_model_warehouse** system catalog.

Syntax

```
PREDICT BY model_name (FEATURES attribute [, attribute] +])
```

Parameter Description

- model_name

  Name of the model of a speculative task.

  Value range: a string. It must comply with the identifier naming convention.

- attribute

  Name of the input feature column of a speculative task.

  Value range: a string. It must comply with the identifier naming convention.

Examples

```
SELECT id, PREDICT BY price_model (FEATURES size,lot), price
FROM houses;
```

Helpful Links

CREATE MODEL and **DROP MODEL**

11.14.156 PREPARE

Function

PREPARE creates a prepared statement.

A prepared statement is a performance optimizing object on the server. When the **PREPARE** statement is executed, the specified query is parsed, analyzed, and rewritten. When **EXECUTE** is executed, the prepared statement is planned and executed. This avoids repetitive parsing and analysis. After the **PREPARE** statement is created, it exists throughout the database session. Once it is created (even if in a transaction block), it will not be deleted when a transaction is rolled back. It can only be deleted by explicitly invoking **DEALLOCATE** or automatically deleted when the session ends.

Precautions

None

Syntax

```
PREPARE name [ ( data_type [, ...] ) ] AS statement;
```

Parameter Description

- **name**

  Specifies the name of a prepared statement. It must be unique in the session.

- **data_type**

  Specifies the type of an argument.

- **statement**

Specifies a **SELECT**, **INSERT**, **UPDATE**, **DELETE**, **MERGE INTO**, or **VALUES** statement.

Examples

See **Examples** in **EXECUTE**.

Helpful Links

DEALLOCATE

11.14.157 PREPARE TRANSACTION

Function

PREPARE TRANSACTION prepares the current transaction for two-phase commit.

After this statement, the transaction is no longer associated with the current session; instead, its state is fully stored on disk, and there is a high probability that it can be committed successfully, even if a database crash occurs before the commit is requested.

Once prepared, a transaction can later be committed or rolled back with **COMMIT PREPARED** or **ROLLBACK PREPARED**, respectively. Those statements can be issued from any session, not only the one that executed the original transaction.

From the point of view of the issuing session, **PREPARE TRANSACTION** is not unlike a **ROLLBACK** statement: after executing it, there is no active current transaction, and the effects of the prepared transaction are no longer visible. (The effects will become visible again if the transaction is committed.)

If the **PREPARE TRANSACTION** statement fails for any reason, it becomes a **ROLLBACK** and the current transaction is canceled.

Precautions

- The transaction function is maintained automatically by the database, and should be not visible to users.
- When running the **PREPARE TRANSACTION** statement, increase the value of **max_prepared_transactions** in configuration file **postgresql.conf**. You are advised to set **max_prepared_transactions** to a value not less than that of **max_connections** so that one pending prepared transaction is available for each session.

Syntax

```
PREPARE TRANSACTION transaction_id;
```

Parameter Description

transaction_id

Specifies an arbitrary identifier that later identifies this transaction for **COMMIT PREPARED** or **ROLLBACK PREPARED**. The identifier must be different from those for current prepared transactions.

Value range: The identifier must be written as a string literal, and must be less than 200 bytes long.

Helpful Links

COMMIT PREPARED and **ROLLBACK PREPARED**

11.14.158 PURGE

Function

The **PURGE** statement can be used to:

- Clear tables or indexes from the recycle bin and release all space related to the objects.
- Clear the recycle bin.
- Clear the objects of a specified tablespace in the recycle bin.

Precautions

- The PURGE operation supports tables (**PURGE TABLE**), indexes (**PURGE INDEX**), and recycle bins (**PURGE RECYCLEBIN**).
- The permission requirements for performing the PURGE operation are as follows:
  - **PURGE TABLE**: The user must be the owner of the table and must have the **USAGE** permission on the schema to which the table belongs. By default, the system administrator has this permission.
  - **PURGE INDEX**: The user must be the owner of the index and have the **USAGE** permission on the schema to which the index belongs. By default, the system administrator has this permission.
  - **PURGE RECYCLEBIN**: Common users can clear only the objects owned by themselves in the recycle bin. In addition, the user must have the **USAGE** permission of the schema to which the objects belong. By default, the system administrator can clear all objects in the recycle bin.

Syntax

```
PURGE { TABLE [schema_name.]table_name
        | INDEX index_name
        | RECYCLEBIN
      }
```

Parameter Description

- *[schema_name.]*

  Schema name
- TABLE *[schema_name.] table_name*

  Clears a specified table in the recycle bin.
- INDEX *index_name*

  Clears a specified index in the recycle bin.

- RECYCLEBIN
  Clears the objects in the recycle bin.

Examples

```
-- Create the reason_table_space tablespace.
openGauss=#  CREATE TABLESPACE REASON_TABLE_SPACE1 owner tpcds RELATIVE location 'tablespace/
tsp_reason1';
-- Create the tpcds.reason_t1 table in the tablespace.
openGauss=#  CREATE TABLE tpcds.reason_t1
 (
  r_reason_sk    integer,
  r_reason_id    character(16),
  r_reason_desc  character(100)
  ) tablespace reason_table_space1;
-- Create the tpcds.reason_t2 table in the tablespace.
openGauss=#  CREATE TABLE tpcds.reason_t2
 (
  r_reason_sk    integer,
  r_reason_id    character(16),
  r_reason_desc  character(100)
  ) tablespace reason_table_space1;
-- Create the tpcds.reason_t3 table in the tablespace.
openGauss=#  CREATE TABLE tpcds.reason_t3
 (
  r_reason_sk    integer,
  r_reason_id    character(16),
  r_reason_desc  character(100)
  ) tablespace reason_table_space1;
-- Create an index on the tpcds.reason_t1 table.
openGauss=#  CREATE INDEX index_t1 on tpcds.reason_t1(r_reason_id);
openGauss=#  DROP TABLE tpcds.reason_t1;
openGauss=#  DROP TABLE tpcds.reason_t2;
openGauss=#  DROP TABLE tpcds.reason_t3;
-- View the recycle bin.
openGauss=#  SELECT rcyname,rcyoriginname,rcytablespace FROM GS_RECYCLEBIN;
        rcyname        | rcyoriginname | rcytablespace
-----------------------+---------------+---------------
 BIN$16409$2CEE988==$0 | reason_t1     |         16408
 BIN$16412$2CF2188==$0 | reason_t2     |         16408
 BIN$16415$2CF2EC8==$0 | reason_t3     |         16408
 BIN$16418$2CF3EC8==$0 | index_t1      |             0
(4 rows)
-- Purge the table.
openGauss=#  PURGE TABLE tpcds.reason_t3;
openGauss=#  SELECT rcyname,rcyoriginname,rcytablespace FROM GS_RECYCLEBIN;
        rcyname        | rcyoriginname | rcytablespace
-----------------------+---------------+---------------
 BIN$16409$2CEE988==$0 | reason_t1     |         16408
 BIN$16412$2CF2188==$0 | reason_t2     |         16408
 BIN$16418$2CF3EC8==$0 | index_t1      |             0
(3 rows)
-- Purge the index.
openGauss=#  PURGE INDEX tindex_t1;
openGauss=#  SELECT rcyname,rcyoriginname,rcytablespace FROM GS_RECYCLEBIN;
        rcyname        | rcyoriginname | rcytablespace
-----------------------+---------------+---------------
 BIN$16409$2CEE988==$0 | reason_t1     |         16408
 BIN$16412$2CF2188==$0 | reason_t2     |         16408
(2 rows)
-- Purge all objects in the recycle bin.
openGauss=#  PURGE recyclebin;
openGauss=#  SELECT rcyname,rcyoriginname,rcytablespace FROM GS_RECYCLEBIN;
        rcyname        | rcyoriginname | rcytablespace
-----------------------+---------------+---------------
(0 rows)
```

11.14.159 REASSIGN OWNED

Function

REASSIGN OWNED changes the owner of the database object.

REASSIGN OWNED requires that the system change owners of all the database objects owned by **old_roles** to **new_role**.

Precautions

- **REASSIGN OWNED** is often executed before role deletion.
- To run the REASSIGN OWNED statement, you must have the permissions of the original and target roles.

Syntax

```
REASSIGN OWNED BY old_role [, ...] TO new_role;
```

Parameter Description

- **old_role**

  Specifies the role name of the old owner.
- **new_role**

  Specifies the role name of the new owner.

Examples

None

11.14.160 REFRESH INCREMENTAL MATERIALIZED VIEW

Function

REFRESH INCREMENTAL MATERIALIZED VIEW refreshes a materialized view in incremental mode.

Precautions

- Incremental refresh supports only fast-refresh materialized views.
- To refresh a materialized view, you must have the SELECT permission on the base table.

Syntax

```
REFRESH INCREMENTAL MATERIALIZED VIEW mv_name;
```

Parameter Description

- **mv_name**

  Name of the materialized view to be refreshed.

Examples

```
-- Create an ordinary table.
openGauss=# CREATE TABLE my_table (c1 int, c2 int);
-- Create a fast-refresh materialized view.
openGauss=# CREATE INCREMENTAL MATERIALIZED VIEW my_imv AS SELECT * FROM my_table;
-- Write data to the base table.
openGauss=# INSERT INTO my_table VALUES(1,1),(2,2);
-- Incrementally refresh the fast-refresh materialized view my_imv.
openGauss=# REFRESH INCREMENTAL MATERIALIZED VIEW my_imv;
```

Helpful Links

ALTER MATERIALIZED VIEW, **CREATE INCREMENTAL MATERIALIZED VIEW**, **CREATE MATERIALIZED VIEW**, **CREATE TABLE**, **DROP MATERIALIZED VIEW**, and **REFRESH MATERIALIZED VIEW**

11.14.161 REFRESH MATERIALIZED VIEW

Function

REFRESH MATERIALIZED VIEW refreshes materialized views in full refresh mode.

Precautions

- Full refreshing can be performed on both full and fast-refresh materialized views.
- To refresh a materialized view, you must have the SELECT permission on the base table.

Syntax

```
REFRESH MATERIALIZED VIEW mv_name;
```

Parameter Description

- **mv_name**

  Name of the materialized view to be refreshed.

Examples

```
-- Create an ordinary table.
openGauss=# CREATE TABLE my_table (c1 int, c2 int);
-- Create a complete-refresh materialized view.
openGauss=# CREATE MATERIALIZED VIEW my_mv AS SELECT * FROM my_table;
-- Create a fast-refresh materialized view.
openGauss=# CREATE INCREMENTAL MATERIALIZED VIEW my_imv AS SELECT * FROM my_table;
-- Write data to the base table.
openGauss=# INSERT INTO my_table VALUES(1,1),(2,2);
-- Refresh the complete-refresh materialized view my_mv.
openGauss=# REFRESH MATERIALIZED VIEW my_mv;
-- Fully refresh the fast-refresh materialized view my_imv.
openGauss=# REFRESH MATERIALIZED VIEW my_imv;
```

Helpful Links

ALTER MATERIALIZED VIEW, **CREATE INCREMENTAL MATERIALIZED VIEW**, **CREATE MATERIALIZED VIEW**, **CREATE TABLE**, **DROP MATERIALIZED VIEW**, and **REFRESH INCREMENTAL MATERIALIZED VIEW**

11.14.162 REINDEX

Function

REINDEX rebuilds an index using the data stored in the index's table, replacing the old copy of the index.

There are several scenarios in which **REINDEX** can be used:

- An index has become corrupted, and no longer contains valid data.
- An index has become "bloated", that is, it contains many empty or nearly-empty pages.
- You have altered a storage parameter (such as a fill factor) for an index, and wish that the change takes full effect.
- An index build with the **CONCURRENTLY** option failed, leaving an "invalid" index.

Precautions

REINDEX DATABASE and **REINDEX SYSTEM** type cannot be performed in transaction blocks. Currently, REINDEX operations cannot be performed on materialized views.

Syntax

- Rebuild a general index.
```
REINDEX { INDEX | [INTERNAL] TABLE | DATABASE | SYSTEM }  name [ FORCE ];
```
- Rebuild an index partition.
```
REINDEX  { INDEX|  [INTERNAL] TABLE}  name
    PARTITION partition_name [ FORCE  ];
```

Parameter Description

- **INDEX**

  Recreates the specified index.
- **INTERNAL TABLE**

  Recreates the Desc table index of a column-store table. The TOAST table (if any) of the table is reindexed as well.
- **TABLE**

  Recreates all indexes of a specified table. If a table has a TOAST table, the table will also be reindexed. If an index in the table has been invalidated by running **alter unusable**, the index cannot be recreated.
- **DATABASE**

  Recreates all indexes within the current database.
- **SYSTEM**

  Recreates all indexes on system catalogs within the current database. Indexes on user tables are not processed.
- **name**

  Specifies the name of the index, table, or database whose index needs to be recreated. Tables and indexes can be schema-qualified.

NOTE

REINDEX DATABASE and **SYSTEM** can create indexes for only the current database. Therefore, **name** must be the same as the current database name.

- **FORCE**

  Specifies an invalid option, which will be ignored.

- **partition_name**

  Specifies the name of the partition or index partition to be recreated.

  Value range:

  - If **REINDEX INDEX** is used, specify the name of an index partition.
  - If it is **REINDEX TABLE**, specify the name of a partition.
  - If it is **REINDEX INTERNAL TABLE**, specify the name of a partition in a column-store partitioned table.

NOTICE

REINDEX DATABASE and **REINDEX SYSTEM** type cannot be performed in transaction blocks.

Examples

```
-- Create a row-store table tpcds.customer_t1 and create an index on the c_customer_sk column in the table.
openGauss=# CREATE TABLE tpcds.customer_t1
(
    c_customer_sk             integer               not null,
    c_customer_id             char(16)              not null,
    c_current_cdemo_sk        integer                       ,
    c_current_hdemo_sk        integer                       ,
    c_current_addr_sk         integer                       ,
    c_first_shipto_date_sk    integer                       ,
    c_first_sales_date_sk     integer                       ,
    c_salutation              char(10)                      ,
    c_first_name              char(20)                      ,
    c_last_name               char(30)                      ,
    c_preferred_cust_flag     char(1)                       ,
    c_birth_day               integer                       ,
    c_birth_month             integer                       ,
    c_birth_year              integer                       ,
    c_birth_country           varchar(20)                   ,
    c_login                   char(13)                      ,
    c_email_address           char(50)                      ,
    c_last_review_date        char(10)
)
WITH (orientation = row);

openGauss=# CREATE INDEX tpcds_customer_index1 ON tpcds.customer_t1 (c_customer_sk);

openGauss=# INSERT INTO tpcds.customer_t1 SELECT * FROM tpcds.customer WHERE c_customer_sk < 10;

-- Rebuild a single index.
openGauss=# REINDEX INDEX tpcds.tpcds_customer_index1;

-- Rebuild all indexes in the tpcds.customer_t1 table:
openGauss=# REINDEX TABLE tpcds.customer_t1;

Delete the tpcds.customer_t1 table.
openGauss=# DROP TABLE tpcds.customer_t1;
```

Suggestions

- INTERNAL TABLE

  This scenario is used for fault recovery. You are not advised to perform concurrent operations.
- DATABASE

  You are not allowed to re-index a database in a transaction.
- SYSTEM

  You are not allowed to re-index system catalogs in transactions.

11.14.163 RELEASE SAVEPOINT

Function

RELEASE SAVEPOINT destroys a savepoint previously defined in the current transaction.

Destroying a savepoint makes it unavailable as a rollback point, but it has no other user visible behavior. It does not undo the effects of statements executed after the savepoint was established. To do that, use **ROLLBACK TO SAVEPOINT**. Destroying a savepoint when it is no longer needed allows the system to recycle some resources earlier than transaction end.

RELEASE SAVEPOINT also destroys all savepoints that were established after the named savepoint was established.

Precautions

- Specifying a savepoint name that was not previously defined causes an error.
- It is not possible to release a savepoint when the transaction is in an aborted state.
- If multiple savepoints have the same name, only the one that was most recently defined is released.

Syntax

```
RELEASE [ SAVEPOINT ] savepoint_name;
```

Parameter Description

savepoint_name

Specifies the name of the savepoint you want to destroy.

Examples

```
-- Create a table.
openGauss=# CREATE TABLE tpcds.table1(a int);

-- Start a transaction.
openGauss=# START TRANSACTION;

-- Insert data.
openGauss=# INSERT INTO tpcds.table1 VALUES (3);
```

```
-- Establish a savepoint.
openGauss=# SAVEPOINT my_savepoint;

-- Insert data.
openGauss=# INSERT INTO tpcds.table1 VALUES (4);

-- Delete the savepoint.
openGauss=# RELEASE SAVEPOINT my_savepoint;

-- Commit the transaction.
openGauss=# COMMIT;

-- Query the table content, which should contain both 3 and 4.
openGauss=# SELECT * FROM tpcds.table1;

-- Delete the table.
openGauss=# DROP TABLE tpcds.table1;
```

Helpful Links

SAVEPOINT and **ROLLBACK TO SAVEPOINT**

11.14.164 RESET

Function

RESET restores run-time parameters to their default values. The default values are parameter default values complied in the **postgresql.conf** configuration file.

RESET is an alternative spelling for:

SET configuration_parameter TO DEFAULT

Precautions

RESET and **SET** have the same transaction behavior. Their impact will be rolled back.

Syntax

```
RESET {configuration_parameter | CURRENT_SCHEMA | TIME ZONE | TRANSACTION ISOLATION LEVEL |
SESSION AUTHORIZATION | ALL };
```

Parameter Description

- **configuration_parameter**

  Specifies the name of a settable run-time parameter.

  Value range: run-time parameters. You can view them by running the **SHOW ALL** statement.

- **CURRENT_SCHEMA**

  Specifies the current schema.

- **TIME ZONE**

  Specifies the time zone.

- **TRANSACTION ISOLATION LEVEL**

  Specifies the transaction isolation level.

- **SESSION AUTHORIZATION**

  Specifies the session authorization.
- **ALL**

  Resets all settable run-time parameters to default values.

Examples

```
-- Reset timezone to the default value.
openGauss=# RESET timezone;

-- Set all parameters to their default values.
openGauss=# RESET ALL;
```

Helpful Links

SET and **SHOW**

11.14.165 REVOKE

Function

REVOKE revokes permissions from one or more roles.

Precautions

If a non-owner user of an object attempts to **REVOKE** permission on the object, the statement is executed based on the following rules:

- If the user has no permissions whatsoever on the object, the statement will fail outright.
- If an authorized user has some permissions, only the permissions with authorization options are revoked.
- If the authorized user does not have the authorization option, the **REVOKE ALL PRIVILEGES** form will issue an error message. For other forms of statements, if the permission specified in the statement does not have the corresponding authorization option, the statement will issue a warning.

Syntax

- Revoke the permission on a specified table or view.
```
REVOKE [ GRANT OPTION FOR ]
    { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | ALTER | DROP | COMMENT |
INDEX | VACUUM }[, ...]
    | ALL [ PRIVILEGES ] }
    ON { [ TABLE ] table_name [, ...]
       | ALL TABLES IN SCHEMA schema_name [, ...] }
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```
- Revoke the permission on a specified field in a table.
```
REVOKE [ GRANT OPTION FOR ]
    { {{ SELECT | INSERT | UPDATE | REFERENCES | COMMENT } ( column_name [, ...] )}[, ...]
    | ALL [ PRIVILEGES ] ( column_name [, ...] ) }
    ON [ TABLE ] table_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- Revoke the permission on a specified sequence. The **LARGE** field is optional. The recycling statement does not distinguish whether the sequence is LARGE.
```
REVOKE [ GRANT OPTION FOR ]
    { { SELECT | UPDATE | ALTER | DROP | COMMENT }[, ...]
    | ALL [ PRIVILEGES ] }
    ON { [ [ LARGE ] SEQUENCE ] sequence_name [, ...]
       | ALL SEQUENCES IN SCHEMA schema_name [, ...] }
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```
- Revoke the permission on a specified database.
```
REVOKE [ GRANT OPTION FOR ]
    { { CREATE | CONNECT | TEMPORARY | TEMP | ALTER | DROP | COMMENT } [, ...]
    | ALL [ PRIVILEGES ] }
    ON DATABASE database_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```
- Revoke the permission on a specified domain.
```
REVOKE [ GRANT OPTION FOR ]
    { USAGE | ALL [ PRIVILEGES ] }
    ON DOMAIN domain_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```
- Revoke the permission on a specified CMK.
```
REVOKE [ GRANT OPTION FOR ]
    { { USAGE | DROP } [, ...] | ALL [PRIVILEGES] }
    ON CLIENT_MASTER_KEYS client_master_keys_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```
- Revoke the permission on a specified CEK.
```
REVOKE [ GRANT OPTION FOR ]
    { { USAGE | DROP } [, ...] | ALL [PRIVILEGES]}
    ON COLUMN_ENCRYPTION_KEYS column_encryption_keys_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```
- Revoke the permission on a specified directory.
```
REVOKE [ GRANT OPTION FOR ]
    { { READ | WRITE | ALTER | DROP } [, ...] | ALL [ PRIVILEGES ] }
    ON DIRECTORY directory_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```
- Revoke the permission on a specified external data source.
```
REVOKE [ GRANT OPTION FOR ]
   { USAGE | ALL [ PRIVILEGES ] }
    ON FOREIGN DATA WRAPPER fdw_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```
- Revoke the permission on a specified external server.
```
REVOKE [ GRANT OPTION FOR ]
    { { USAGE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON FOREIGN SERVER server_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```
- Revoke the permission on a specified function.
```
REVOKE [ GRANT OPTION FOR ]
    { { EXECUTE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON { FUNCTION {function_name ( [ {[ argmode ] [ arg_name ] arg_type} [, ...] ] )} [, ...]
       | ALL FUNCTIONS IN SCHEMA schema_name [, ...] }
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```
- Revoke the permission on a specified stored procedure.
```
REVOKE [ GRANT OPTION FOR ]
    { { EXECUTE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON { PROCEDURE {proc_name ( [ {[ argmode ] [ arg_name ] arg_type} [, ...] ] )} [, ...]
```

```
      | ALL PROCEDURE IN SCHEMA schema_name [, ...] }
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- Revoke the permission on a specified procedural language.

```
REVOKE [ GRANT OPTION FOR ]
   { USAGE | ALL [ PRIVILEGES ] }
    ON LANGUAGE lang_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- Revoke the permission on a specified large object.

```
REVOKE [ GRANT OPTION FOR ]
    { { SELECT | UPDATE } [, ...] | ALL [ PRIVILEGES ] }
    ON LARGE OBJECT loid [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- Revoke the permission on a specified schema.

```
REVOKE [ GRANT OPTION FOR ]
    { { CREATE | USAGE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON SCHEMA schema_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- Revoke the permission on a specified tablespace.

```
REVOKE [ GRANT OPTION FOR ]
    { { CREATE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON TABLESPACE tablespace_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- Revoke the permission on a specified type.

```
REVOKE [ GRANT OPTION FOR ]
   { { USAGE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON TYPE type_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- Revoke the permission on a data source object.

```
REVOKE [ GRANT OPTION FOR ]
   { USAGE | ALL [PRIVILEGES] }
    ON DATA SOURCE src_name [, ...]
    FROM {[GROUP] role_name | PUBLIC} [, ...]
   [ CASCADE | RESTRICT ];
```

- Revoke the permission on a package object.

```
REVOKE [ GRANT OPTION FOR ]
   { { EXECUTE | ALTER | DROP | COMMENT } [, ...] | ALL [PRIVILEGES] }
   ON PACKAGE package_name [, ...]
   FROM {[GROUP] role_name | PUBLIC} [, ...]
   [ CASCADE | RESTRICT ];
```

- Revoke permissions from a role.

```
REVOKE [ ADMIN OPTION FOR ]
    role_name [, ...] FROM role_name [, ...]
    [ CASCADE | RESTRICT ];
```

- Revoke the sysadmin permission on a role.

```
REVOKE ALL { PRIVILEGES | PRIVILEGE } FROM role_name;
```

- Revoke the ANY permissions.

```
REVOKE [ ADMIN OPTION FOR ]
  { CREATE ANY TABLE | ALTER ANY TABLE | DROP ANY TABLE | SELECT ANY TABLE | INSERT ANY
TABLE | UPDATE ANY TABLE |
  DELETE ANY TABLE | CREATE ANY SEQUENCE | CREATE ANY INDEX | CREATE ANY FUNCTION |
EXECUTE ANY FUNCTION |
  CREATE ANY PACKAGE | EXECUTE ANY PACKAGE | CREATE ANY TYPE } [, ...]
  FROM [ GROUP ] role_name [, ...];
```

Parameter Description

The keyword **PUBLIC** indicates an implicitly defined group that has all roles.

For details about permission types and parameters, see **Parameter Description** in section "GRANT".

Permissions of a role include the permissions directly granted to the role, permissions inherited from the parent role, and permissions granted to PUBLIC users. Therefore, revoking the SELECT permission on an object from **PUBLIC** users does not necessarily mean that the SELECT permission on the object has been revoked from all roles, because the SELECT permission directly granted to roles and inherited from parent roles remains. Similarly, if the SELECT permission is revoked from a user but is not revoked from PUBLIC users, the user can still run the **SELECT** statement.

If **GRANT OPTION FOR** is specified, the permission cannot be granted to others, but permission itself is not revoked.

If user A holds the UPDATE permission on a table and the **WITH GRANT OPTION** and has granted them to user B, the permission that user B holds is called dependent permission. When user A's permission or grant option is revoked, **CASCADE** must be declared to revoke all dependent permissions.

A user can only revoke permissions that were granted directly by that user. For example, if user A has granted permission with grant option (**WITH ADMIN OPTION**) to user B, and user B has in turn granted it to user C, then user A cannot revoke the permission directly from C. However, user A can revoke the grant option held by user B and use **CASCADE**. In this way, the permission of user C is automatically revoked. For another example, if both user A and user B have granted the same permission to C, A can revoke his own grant but not B's grant, so C will still effectively have the permission.

If the role executing **REVOKE** holds permissions indirectly via more than one role membership path, it is unspecified which containing role will be used to execute the statement. In such cases, it is best practice to use **SET ROLE** to become the specific role you want to do the **REVOKE** as, and then execute **REVOKE**. Failure to do so may lead to deleting permissions not intended to delete, or not deleting any permissions at all.

Examples

See **Examples** in section "GRANT".

Helpful Links

GRANT

11.14.166 ROLLBACK

Function

ROLLBACK rolls back the current transaction and backs out all updates in the transaction.

ROLLBACK backs out of all changes that a transaction makes to a database if the transaction fails to be executed due to a fault.

Precautions

If a **ROLLBACK** statement is executed out of a transaction, no error occurs, but a notice is displayed.

Syntax

```
ROLLBACK [ WORK | TRANSACTION ];
```

Parameter Description

WORK | TRANSACTION

Specifies the optional keyword. that more clearly illustrates the syntax.

Examples

```
-- Start a transaction.
openGauss=# START TRANSACTION;

-- Back out all changes.
openGauss=# ROLLBACK;
```

Helpful Links

COMMIT | END

11.14.167 ROLLBACK PREPARED

Function

ROLLBACK PREPARED cancels a transaction ready for two-phase committing.

Precautions

- The function is only available in maintenance mode (when GUC parameter **xc_maintenance_mode** is **on**). Exercise caution when enabling the mode. It is used by maintenance engineers for troubleshooting. Common users should not use the mode.
- Only the user that initiates a transaction or the system administrator can roll back the transaction.
- The transaction function is maintained automatically by the database, and should be not visible to users.

Syntax

```
ROLLBACK PREPARED transaction_id ;
```

Parameter Description

transaction_id

Specifies the identifier of the transaction to be committed. The identifier must be different from those for current prepared transactions.

Helpful Links

COMMIT PREPARED and **PREPARE TRANSACTION**

11.14.168 ROLLBACK TO SAVEPOINT

Function

ROLLBACK TO SAVEPOINT rolls back to a savepoint. It implicitly destroys all savepoints that were established after the named savepoint.

Rolls back all statements that were executed after the savepoint was established. The savepoint remains valid and can be rolled back to again later, if needed.

Precautions

- Specifying a savepoint name that has not been established is an error.
- Cursors have somewhat non-transactional behavior with respect to savepoints. Any cursor that is opened inside a savepoint will be closed when the savepoint is rolled back. If a previously opened cursor is affected by a **FETCH** or **MOVE** statement inside a savepoint that is later rolled back, the cursor remains at the position that **FETCH** left it pointing to (that is, the cursor motion caused by **FETCH** is not rolled back). Closing a cursor is not undone by rolling back, either. A cursor whose execution causes a transaction to abort is put in a cannot-execute state, so while the transaction can be restored using **ROLLBACK TO SAVEPOINT**, the cursor can no longer be used.
- Use **ROLLBACK TO SAVEPOINT** to roll back to a savepoint. Use **RELEASE SAVEPOINT** to destroy a savepoint but keep the effects of the statements executed after the savepoint was established.

Syntax

```
ROLLBACK [ WORK | TRANSACTION ] TO [ SAVEPOINT ] savepoint_name;
```

Parameter Description

savepoint_name

Rolls back to a savepoint.

Examples

```
-- Undo the effects of the statements executed after my_savepoint was established:
openGauss=# START TRANSACTION;
openGauss=# SAVEPOINT my_savepoint;
openGauss=# ROLLBACK TO SAVEPOINT my_savepoint;
-- Cursor positions are not affected by savepoint rollback.
openGauss=# DECLARE foo CURSOR FOR SELECT 1 UNION SELECT 2;
openGauss=# SAVEPOINT foo;
openGauss=# FETCH 1 FROM foo;
 ?column?
----------
        1
openGauss=# ROLLBACK TO SAVEPOINT foo;
```

```
openGauss=# FETCH 1 FROM foo;
 ?column? 
----------
        2
openGauss=# RELEASE SAVEPOINT my_savepoint;
openGauss=# COMMIT;
```

Helpful Links

SAVEPOINT and **RELEASE SAVEPOINT**

11.14.169 SAVEPOINT

Function

SAVEPOINT establishes a new savepoint within the current transaction.

A savepoint is a special mark inside a transaction. It allows all statements that are executed after it was established to be rolled back, restoring the transaction state to what it was at the time of the savepoint.

Precautions

- Use **ROLLBACK TO SAVEPOINT** to roll back to a savepoint. Use **RELEASE SAVEPOINT** to destroy a savepoint but keep the effects of the statements executed after the savepoint was established.
- Savepoints can only be established when inside a transaction block. Multiple savepoints can be defined in a transaction.
- In the case of an unexpected termination of a thread or process caused by a node or connection failure, or of an error caused by the inconsistency between source and destination table structures in a **COPY FROM** operation, the transaction cannot be rolled back to the established savepoint. Instead, the entire transaction will be rolled back.
- According to the SQL standard, when a savepoint with the same name is created, the previous savepoint with the same name is automatically deleted. In GaussDB, the old savepoint is retained, but only the latest one is used during rollback or release. Releasing the newer savepoint with **RELEASE SAVEPOINT** will cause the older one to again become accessible to **ROLLBACK TO SAVEPOINT** and **RELEASE SAVEPOINT**. In addition, **SAVEPOINT** fully complies with the SQL standard.

Syntax

```
SAVEPOINT savepoint_name;
```

Parameter Description

savepoint_name

Specifies the name of the new savepoint.

Examples

```
-- Create a table.
openGauss=# CREATE TABLE table1(a int);
```

```
-- Start a transaction.
openGauss=# START TRANSACTION;

-- Insert data.
openGauss=# INSERT INTO table1 VALUES (1);

-- Create a savepoint.
openGauss=# SAVEPOINT my_savepoint;

-- Insert data.
openGauss=# INSERT INTO table1 VALUES (2);

-- Roll back the savepoint.
openGauss=# ROLLBACK TO SAVEPOINT my_savepoint;

-- Insert data.
openGauss=# INSERT INTO table1 VALUES (3);

-- Commit the transaction.
openGauss=# COMMIT;

-- Query the content of the table. You can see 1 and 3 at the same time, but cannot see 2 because 2 is
rolled back.
openGauss=# SELECT * FROM table1;

-- Delete the table.
openGauss=# DROP TABLE table1;

-- Create a table.
openGauss=# CREATE TABLE table2(a int);

-- Start a transaction.
openGauss=# START TRANSACTION;

-- Insert data.
openGauss=# INSERT INTO table2 VALUES (3);

-- Create a savepoint.
openGauss=# SAVEPOINT my_savepoint;

-- Insert data.
openGauss=# INSERT INTO table2 VALUES (4);

-- Roll back the savepoint.
openGauss=# RELEASE SAVEPOINT my_savepoint;

-- Commit the transaction.
openGauss=# COMMIT;

-- Query the table content. You can see 3 and 4 at the same time.
openGauss=# SELECT * FROM table2;

-- Delete the table.
openGauss=# DROP TABLE table2;
```

Helpful Links

RELEASE SAVEPOINT and **ROLLBACK TO SAVEPOINT**

11.14.170 SELECT

Function

SELECT retrieves data from a table or view.

Serving as an overlaid filter for a database table, **SELECT** filters required data from the table using SQL keywords.

Precautions

- The owner of a table, users granted with the SELECT permission on the table, or users granted with the SELECT ANY TABLE permission can read data in the table or view. The system administrator has the permission to read data in the table or view by default.
- You must have the SELECT permission on each field used in the SELECT statement.
- Using FOR UPDATE, FOR NO KEY UPDATE, FOR SHARE, or FOR KEY SHARE also requires the UPDATE permission.

Syntax

- Query data.

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
SELECT [/*+ plan_hint */] [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]
{ * | {expression [ [ AS ] output_name ]} [, ...] }
[ FROM from_item [, ...] ]
[ WHERE condition ]
[ [ START WITH condition ] CONNECT BY [NOCYCLE] condition [ ORDER SIBLINGS BY expression ] ]
[ GROUP BY grouping_element [, ...] ]
[ HAVING condition [, ...] ]
[ WINDOW {window_name AS ( window_definition )} [, ...] ]
[ { UNION | INTERSECT | EXCEPT | MINUS } [ ALL | DISTINCT ] select ]
[ ORDER BY {expression [ [ ASC | DESC | USING operator ] | nlssort_expression_clause ] [ NULLS { FIRST |
LAST } ]} [, ...] ]
[ LIMIT { [offset,] count | ALL } ]
[ OFFSET start [ ROW | ROWS ] ]
[ FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY ]
[ {FOR { UPDATE | NO KEY UPDATE | SHARE | KEY SHARE } [ OF table_name [, ...] ] [ NOWAIT | WAIT N ]}
[...] ];
```

NOTE

In condition and expression, you can use the aliases of expressions in **targetlist** in compliance with the following rules:

- Reference only within the same level.
- Only reference aliases in **targetlist**.
- Reference a prior expression in a subsequent expression.
- The **volatile** function cannot be used.
- The **Window** function cannot be used.
- Aliases cannot be referenced in the **join on** condition.
- An error is reported if **targetlist** contains multiple referenced aliases.

- The subquery **with_query** is as follows:

```
with_query_name [ ( column_name [, ...] ) ]
    AS [ [ NOT ] MATERIALIZED ] ( {select | values | insert | update | delete} )
```

- The specified query source **from_item** is as follows:

```
{[ ONLY ] table_name [ * ] [ partition_clause ] [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
[ TABLESAMPLE sampling_method ( argument [, ...] ) [ REPEATABLE ( seed ) ] ]
[ TIMECAPSULE {TIMESTAMP | CSN} expression ]
|( select ) [ AS ] alias [ ( column_alias [, ...] ) ]
|with_query_name [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
|function_name ( [ argument [, ...] ] ) [ AS ] alias [ ( column_alias [, ...] | column_definition [, ...] ) ]
|function_name ( [ argument [, ...] ] ) AS ( column_definition [, ...] )
|from_item [ NATURAL ] join_type from_item [ ON join_condition | USING ( join_column [, ...] ) ]}
```

- The **group** clause is as follows:

```
( )
| expression
| ( expression [, ...] )
| ROLLUP ( { expression | ( expression [, ...] ) } [, ...] )
| CUBE ( { expression | ( expression [, ...] ) } [, ...] )
| GROUPING SETS ( grouping_element [, ...] )
```

- The specified partition **partition_clause** is as follows:

```
PARTITION { ( partition_name ) | FOR (  partition_value [, ...] ) } |
SUBPARTITION { ( subpartition_name ) | FOR (  subpartition_value [, ...] )}
```

  NOTE

  The specified partition applies only to partitioned tables.

- The sorting order **nlssort_expression_clause** is as follows:

```
NLSSORT ( column_name, ' NLS_SORT = { SCHINESE_PINYIN_M | generic_m_ci } ' )
The second parameter can be generic_m_ci, which supports only the case-insensitive order for English characters.
```

- Simplified query syntax, equivalent to **select * from table_name**.

```
TABLE { ONLY {(table_name)| table_name} | table_name [ * ]};
```

Parameter Description

- **WITH [RECURSIVE] with_query [, ...]**

  Specifies one or more subqueries that can be referenced by name in the main query, which is equivalent to a temporary table.

  If **RECURSIVE** is specified, it allows a **SELECT** subquery to reference itself by name.

  The detailed format of **with_query** is as follows: **with_query_name [(column_name [, ...])] AS [[NOT] MATERIALIZED] ({select | values | insert | update | delete})**

  – **with_query_name** specifies the name of the result set generated by a subquery. Such names can be used to access the result sets of subqueries in a query.

  – **column_name** specifies the column name displayed in the subquery result set.

  – Each subquery can be a **SELECT**, **VALUES**, **INSERT**, **UPDATE** or **DELETE** statement.

  – **RECURSIVE can appear only after WITH. In the case of multiple CTEs, you only need to declare RECURSIVE at the first CTE**.

  – You can use **MATERIALIZED** or **NOT MATERIALIZED** to modify the CTE.

    ■ If **MATERIALIZED** is specified, the WITH query will be materialized, and a copy of the subquery result set is generated. The copy is directly queried at the reference point. Therefore, the WITH subquery cannot be jointly optimized with the SELECT statement trunk (for example, predicate pushdown and equivalence class transfer). In this scenario, you can use **NOT MATERIALIZED** for modification. If the WITH query can be executed as a subquery inline, the preceding optimization can be performed.

    ■ If the user does not explicitly declare the materialized attribute, comply with the following rules: If the CTE is referenced only once in the SELECT statement trunk to which it belongs and semantically supports inline execution, it will be rewritten as subquery inline execution. Otherwise, the materialized execution will be performed in CTE Scan mode.

- **plan_hint** clause

  Follows the **SELECT** keyword in the **/*+<*Plan hint*> */** format. It is used to optimize the plan of a **SELECT** statement block. For details, see **Hint-based Tuning**. In each statement, only the first **/*+** *plan_hint* ***/** comment block takes effect as a hint. Multiple hints can be written.

- **ALL**

  Specifies that all rows that meet the conditions are returned. This is the default behavior and can be omitted.

- **DISTINCT [ON (expression [, ...])]**

  Removes all duplicate rows from the **SELECT** result set so one row is kept from each group of duplicates.

  Retains only the first row in the set of rows that have the same result calculated on the given expression.

  NOTICE

  DISTINCT ON expression is explained with the same rule of **ORDER BY**. Unless you use **ORDER BY** to guarantee that the required row appears first, you cannot know what the first row is.

- **SELECT list**

  Specifies the name of a column in the table to be queried. The value can be a part of the column name or all of the column names. The wildcard (*) is used to represent the column name.

  You may use the **AS output_name** clause to give an alias for an output column. The alias is used for the displaying of the output column. The name, value, and type keywords can be used as column aliases.

  Column names can be expressed in the following formats:

  – Manually input column names which are spaced using commas (,).

  – Columns computed in the **FROM** clause.

- **FROM clause**

  Specifies one or more source tables for **SELECT**.

  The **FROM** clause can contain the following elements:

  – table_name

    Specifies the name of a table or view. The schema name can be added before the table name or view name, for example, schema_name.table_name.

  – alias

    Gives a temporary alias to a table to facilitate the quotation by other queries.

    An alias is used for brevity or to eliminate ambiguity for self-joins. If an alias is provided, it completely hides the actual name of the table.

  – TABLESAMPLE *sampling_method* (*argument* [, ...]) [REPEATABLE (*seed*)]

The **TABLESAMPLE** clause following *table_name* specifies that the specified *sampling_method* should be used to retrieve the subset of rows in the table.

The optional **REPEATABLE** clause specifies the number of seeds used to generate random numbers in the sampling method. The seed value can be any non-null constant value. If the table was not changed during the query, the two queries having the same seed and *argument* values will select the same sampling in this table. However, different seed values usually generate different samples. If **REPEATABLE** is not specified, a new random sample will be selected for each query based on the seed generated by the system.

- TIMECAPSULE { TIMESTAMP | CSN } expression

  Queries the table data of a specified CSN or at a specified time point.

  Currently, the following tables do not support flashback query: system catalogs, column-store tables, memory tables, DFS tables, global temporary tables, local temporary tables, unlogged tables, views, sequence tables, hash bucket tables, shared tables, inherited tables, and tables with the **PARTIAL CLUSTER KEY** constraint.

  - TIMECAPSULE TIMESTAMP

    Searches for the result set of a specified time point based on the date as the flashback query flag. *date* must be a valid past timestamp

  - TIMECAPSULE CSN

    Searches for the result set of a specified CSN point based on the CSN flashback of the table as the flashback query flag. The CSN can be obtained from **snpcsn** recorded in **gs_txn_snapshot**.

    NOTE

    - A flashback query cannot span statements that affect the table structure or physical storage. Otherwise, an error is reported. Between the flashback point and the current point, if a statement (TRUNCATE, DDL, DCL, or VACUUM FULL) has been executed to modify the table structure or affect physical storage, the flashback fails.
    - Flashback query does not support index query. Flashback query supports only seqScan for full table scanning.
    - When the flashback point is too old, the old version cannot be obtained because the flashback version is recycled. As a result, the flashback fails and the error message "Restore point too old" is displayed.
    - The flashback point is specified by time. The maximum difference between the flashback point and the actual time is 3 seconds.
    - After truncating a table, perform a flashback query or flashback on the table. The error message "Snapshot too old" is displayed when a flashback is performed at a specified time point. Data cannot be found or the error message "Snapshot too old" is reported during the CSN-based flashback.

- column_alias

  Specifies the column alias.

- PARTITION

  Queries data in the specified partition in a partitioned table.

- partition_name

  Specifies the name of a partition.

- partition_value

  Specifies the value of the specified partition key. If there are many partition keys, use the **PARTITION FOR** clause to specify the value of the only partition key you want to use.

- SUBPARTITION

  Queries data in the specified level-2 partition in a partitioned table.

- subpartition_name

  Specifies the name of a level-2 partition name.

- subpartition_value

  Specifies the key values of specified level-1 and level-2 partitions. The values of the two partition keys specified by the **SUBPARTITION FOR** clause uniquely identify a level-2 partition.

- subquery

  Performs a subquery in the **FROM** clause. A temporary table is created to save subquery results.

- with_query_name

  Specifies that the **WITH** clause can also be used as the source of the **FROM** clause and can be referenced by the name of the **WITH** query.

- function_name

  Function name. Function calls can appear in the **FROM** clause.

- join_type

  The options are as follows:

  - [INNER] JOIN

    A **JOIN** clause combines two **FROM** items. You can use parentheses to determine the order of nesting. In the absence of parentheses, **JOIN** nests left-to-right.

    In any case, **JOIN** binds more tightly than the commas separating **FROM** items.

  - LEFT [OUTER] JOIN

    Returns all rows that meet join conditions in the Cartesian product, plus those rows that do not match the right table rows in the left table by join conditions. This left-hand row is extended to the full width of the joined table by inserting **NULL** values for the right-hand columns. Note that only the **JOIN** clause's own condition is considered while the system decides which rows have matches. Outer conditions are applied afterward.

  - RIGHT [OUTER] JOIN

    Returns all the joined rows, plus one row for each unmatched right-hand row (extended with **NULL** on the left).

    This is just a notational convenience, since you could convert it to a **LEFT OUTER JOIN** by switching the left and right inputs.

- FULL [OUTER] JOIN

  Returns all the joined rows, pluses one row for each unmatched left-hand row (extended with **NULL** on the right), and pluses one row for each unmatched right-hand row (extended with **NULL** on the left).

- CROSS JOIN

  Is equivalent to **INNER JOIN ON (TRUE)**, which means no rows are removed by qualification. These join types are just a notational convenience, since they do nothing you could not do with plain **FROM** and **WHERE**.

  NOTE

  For the **INNER** and **OUTER** join types, a join condition must be specified, namely exactly one of **NATURAL ON**, **join_condition**, or **USING (join_column [, ...])**. For **CROSS JOIN**, none of these clauses can appear.

CROSS JOIN and **INNER JOIN** produce a simple Cartesian product, the same result as you get from listing the two items at the top level of **FROM**.

– ON join_condition

  Defines which rows have matches in joins. Example: ON left_table.a = right_table.a You are not advised to use numeric types such as int for **join_condition**, because such types can be implicitly converted to bool values (non-zero values are implicitly converted to **true** and **0** is implicitly converted to **false**), which may cause unexpected results.

– USING(join_column[, ...])

  Abbreviation of **ON left_table.a = right_table.a AND left_table.b = right_table.b ....**. The names of the corresponding columns must be the same.

– NATURAL

  Is a shorthand for a **USING** list that mentions all columns in the two tables that have the same names.

– from item

  Specifies the name of the query source object connected.

- **WHERE clause**

  Forms an expression for row selection to narrow down the query range of **SELECT**. **condition** indicates any expression that returns a value of Boolean type. Rows that do not meet this condition will not be retrieved. You are not advised to use numeric types such as int for **condition**, because such types can be implicitly converted to bool values (non-zero values are implicitly converted to **true** and **0** is implicitly converted to **false**), which may cause unexpected results.

  In the **WHERE** clause, you can use the operator (+) to convert a table join to an outer join. However, this method is not recommended because it is not the standard SQL syntax and may raise syntax compatibility issues during platform migration. There are many restrictions on using the operator (+):

  a. It can appear only in the **WHERE** clause.

  b. If a table join has been specified in the **FROM** clause, the operator (+) cannot be used in the **WHERE** clause.

c. The operator (+) can work only on columns of tables or views, instead of on expressions.
d. If table A and table B have multiple join conditions, the operator (+) must be specified in all the conditions. Otherwise, the operator (+) will not take effect, and the table join will be converted into an inner join without any prompt information.
e. Tables specified in a join condition where the operator (+) works cannot cross queries or subqueries. If tables where the operator (+) works are not in the **FROM** clause of the current query or subquery, an error will be reported. If a peer table for the operator (+) does not exist, no error will be reported and the table join will be converted into an inner join.
    i. Expressions where the operator (+) is used cannot be directly connected through **OR**.
f. If a column where the operator (+) works is compared with a constant, the expression becomes a part of the join condition.
g. A table cannot have multiple foreign tables.
h. The operator (+) can appear only in the following expressions: comparison, NOT, ANY, ALL, IN, NULLIF, IS DISTINCT FROM, and IS OF. It is not allowed in other types of expressions. In addition, these expressions cannot be connected through **AND** or **OR**.
i. The operator (+) can be used to convert a table join only to a left or right outer join, instead of a full join. That is, the operator (+) cannot be specified on both tables of an expression.

NOTICE

For the **WHERE** clause, if special character **%**, _, or **** is queried in **LIKE**, add the slash **** before each character.

- **START WITH clause**

  The **START WITH** clause is usually used together with the **CONNECT BY** clause and indicates the initial condition of recursion. Data is traversed recursively and hierarchically. If this clause is omitted and the **CONNECT BY** clause is used alone, all rows in the table are used as the initial set. For details, see **CONNECT BY**.
- **CONNECT BY clause**

  CONNECT BY indicates a recursive join condition. It is used together with START WITH to implement data traversal and recursion. For example:

```
openGauss=#  create table test(name varchar, id int, fatherid int);
openGauss=#  insert into test values('A', 1, 0), ('B', 2, 1),('C',3,1),('D',4,1),('E',5,2);
openGauss=#  select * from test start with id = 1 connect by prior id = fatherid order siblings by id
desc;
 name | id | fatherid
------+----+----------
 A    |  1 |        0
 D    |  4 |        1
 C    |  3 |        1
 B    |  2 |        1
 E    |  5 |        2
(5 rows)
```

  In the CONNECT BY condition, the PRIOR keyword can be specified for a column to indicate that the column is recursive. If NOCYCLE is added before the recursive join condition, recursion stops when a loop record is encountered. (Note: A SELECT statement containing the START WITH .. CONNECT BY clause does not support the FOR SHARE or UPDATE lock.)

The process of executing the START WITH statement is as follows:

a. The initial data set is selected based on the condition in the START WITH clause. In the preceding example, **('A', 1, 0)** is selected first. Then, this initial data set is set as the working set.

b. If the working set is not empty, the data in the working set is used as the input for the next query. The filter criteria are specified in the CONNECT BY clause. The keyword PRIOR indicates the current record. For example, **prior id = fatherid** in the preceding example indicates that the ID of the current record is the **fatherid** of the next record.

c. Set the data set filtered in step 2 as the working set and repeat the operation in step 2.

In addition, the database adds the following pseudocolumns to each selected data record so that users can learn about the location of the data in the recursive or tree structure.

– **LEVEL**: node level.

– **CONNECT_BY_ISLEAF**: specifies whether a node is a leaf node.

In addition to pseudocolumns, the following query functions are provided (for details, see **Hierarchical Recursion Query Functions**):

– **sys_connect_by_path(col, separtor)**: returns the connection path from the root node to the current row. The **col** parameter indicates the name of the column displayed in the path, and the **separator** parameter indicates the connector.

– **connect_by_root(col)**: displays the top-level node of the node. **col** indicates the name of the output column.

If a loop exists in the data set, the database provides loop detection. By default, if a loop is detected, an error is reported and no data is returned. In addition, the NOCYCLE keyword is provided. With it, the query can be executed normally and when the first duplicate data record is found, the query exits directly instead of reporting an error.

Besides, in the hierarchical query, the search is performed strictly according to the depth-first order. If ROWNUM is used as the filtering condition in START WITH or CONNECT BY, the value of ROWNUM is increased by 1 for each record to be returned. Then, the record is verified based on ROWNUM conditions. Records that do not satisfy the conditions are discarded and the value of ROWNUM is decreased by 1.

NOTICE

- The PRIOR keyword can be used only in the CONNECT BY clause instead of the START WITH clause.
- The PRIOR keyword can be specified only for columns in the table instead of expressions, pseudocolumns, or type conversion. For example, PRIOR (a + 1) is not allowed.
- In the CONNECT BY clause, the column using the keyword PRIOR cannot be in the same condition with pseudocolumns such as level and rownum, but they can be in different conditions. For example, (PRIOR a = level) is not allowed, but (PRIOR a = b) and (level = 1) is allowed. Different conditions refer to the conditions connected by AND at the top of the CONNECT BY clause. For example, (PRIOR a = 1 or level = 1) is considered as a condition and is not allowed.
- In the START WITH and CONNECT BY clauses, pseudocolumns can not be used for sublinks, for example, "rownum = (subquery)" or "rownum in (subquery)".
- When START WITH and CONNECT BY are invoked on the CTE defined by WITH AS, if there are multiple CTEs, ensure that the definition of each CTE does not depend on other CTEs.
- If no loop exists in the data but the error message "runs into cycle" is reported, increase the value of **max_recursive_times**.
- Optimization suggestions for the START WITH clause:
  - Create indexes based on the conditions in the CONNECT BY clause to improve the performance of the START WITH clause.
  - Identify bottlenecks based on the plan collected by running EXPLAIN PERFORMANCE or in WDR. If the recursive operator (inner plan) of RECURSIVE UNION is the Hash Join operator, but the hash table is created for the temporary table **tmp_result** or the hash table in plan is materialized (that is, the batch size is greater than 1), the possible cause is that the value of **work_mem** is too small. As a result, the hash table cannot be created for the outer data table. You can increase the value of **work_mem** to improve performance.

    Note: GaussDB optimizes tables with a small volume of data and caches table results in hash tables to improve performance. In this case, indexes are not required. However, if the data volume exceeds the limit specified by **work_mem**, the optimization becomes invalid. In this case, you can create indexes for optimization.

- **ORDER SIBLINGS BY** clause

  The output of the START WITH statement is returned level by level. However, there is no sequence guarantee at each level because the database automatically selects the optimal execution path during each round of query. In the preceding example, A is output first, but the sequence of B, C, and D is not fixed. If you have requirements on the final output sequence, you can use ORDER SIBLINGS BY. The usage of ORDER SIBLINGS BY is the same as that of ORDER BY. ORDER SIBLINGS BY is used for sorting at each level during recursion.

NOTICE

ORDER SIBLINGS BY only supports sorting by adding column names instead of invoking system functions for column names.

- **GROUP BY clause**

  Condenses query results into a single row all selected rows that share the same values for the grouped expressions.

  – CUBE ({ expression | (expression [, ...]) } [, ...])

    A CUBE grouping is an extension to the **GROUP BY** clause that creates subtotals for all of the possible combinations of the given list of grouping columns (or expressions). In terms of multidimensional analysis, CUBE generates all the subtotals that could be calculated for a data cube with the specified dimensions. For example, given three expressions (n=3) in the CUBE clause, the operation results in $2^n = 2^3 = 8$ groupings. Rows grouped on the values of *n* expressions are called regular rows, and the rest are called superaggregate rows.

  – GROUPING SETS (grouping_element [, ...])

    Another extension to the **GROUP BY** clause. It allows users to specify multiple **GROUP BY** clauses. This improves efficiency by trimming away unnecessary data. After you specify the set of groups that you want to create using a **GROUPING SETS** expression within a **GROUP BY** clause, the database does not need to compute a whole **ROLLUP** or **CUBE**.

NOTICE

If the **SELECT** list expression quotes some ungrouped fields and no aggregate function is used, an error is displayed. This is because multiple values may be returned for ungrouped fields.

- **HAVING clause**

  Selects special groups by working with the **GROUP BY** clause. The **HAVING** clause compares some attributes of groups with a constant. Only groups that matching the logical expression in the **HAVING** clause are extracted.

- **WINDOW clause**

  The general format is **WINDOW window_name AS (window_definition) [, ...]**. **window_name** is a name can be referenced by **window_definition**. **window_definition** can be expressed in the following forms:

```
[ existing_window_name ]
[ PARTITION BY expression [, ...] ]
[ ORDER BY expression [ ASC | DESC | USING operator ] [ NULLS { FIRST | LAST } ] [, ...] ]
[ frame_clause ]
```

  frame_clause defines a **window frame** for the window function. The window function (not all window functions) depends on **window frame** and **window frame** is a set of relevant rows of the current query row. **frame_clause** can be expressed in the following forms:

```
[ RANGE | ROWS ] frame_start
[ RANGE | ROWS ] BETWEEN frame_start AND frame_end
```

  frame_start and **frame_end** can be expressed in the following forms:

```
UNBOUNDED PRECEDING
value PRECEDING
CURRENT ROW
value FOLLOWING
UNBOUNDED FOLLOWING
```

NOTICE

For the query of column storage table, only **row_number** window function is supported, and **frame_clause** is not supported.

- **UNION clause**

  Computes the set union of the rows returned by the involved **SELECT** statements.

  The **UNION** clause has the following constraints:

  - By default, the result of **UNION** does not contain any duplicate rows unless the **ALL** clause is declared.
  - Multiple **UNION** operators in the same SELECT statement are evaluated left to right, unless otherwise specified by parentheses.
  - **FOR UPDATE**, **FOR NO KEY UPDATE**, **FOR SHARE**, and **FOR KEY SHARE** cannot be specified in the result or input of **UNION**.

  General expression:

  select_statement UNION [ALL] select_statement

  - **select_statement** can be any SELECT statement without the **ORDER BY**, **LIMIT**, **FOR UPDATE**, **FOR NO KEY UPDATE**, **FOR SHARE**, or **FOR KEY SHARE** clause.
  - **ORDER BY** and **LIMIT** can be attached to the subexpression if it is enclosed in parentheses.

- **INTERSECT clause**

  Computes the set intersection of rows returned by the involved **SELECT** statements. The result of **INTERSECT** does not contain any duplicate rows.

  The **INTERSECT** clause has the following constraints:

  - Multiple **INTERSECT** operators in the same **SELECT** statement are evaluated left to right, unless otherwise specified by parentheses.
  - Processing **INTERSECT** preferentially when **UNION** and **INTERSECT** operations are executed for results of multiple **SELECT** statements.

  General format:

  select_statement INTERSECT select_statement

  select_statement can be any SELECT statement without the **FOR UPDATE**, **FOR NO KEY UPDATE**, **FOR SHARE**, or **FOR KEY SHARE** clause.

- **EXCEPT clause**

  Has the following common form:

  select_statement EXCEPT [ALL] select_statement

  select_statement can be any SELECT statement without the **FOR UPDATE**, **FOR NO KEY UPDATE**, **FOR SHARE**, or **FOR KEY SHARE** clause.

  The **EXCEPT** operator computes the set of rows that are in the result of the left SELECT statement but not in the result of the right one.

The result of **EXCEPT** does not contain any duplicate rows unless the **ALL** clause is declared. To execute **ALL**, a row that has *m* duplicates in the left table and *n* duplicates in the right table will appear MAX(*m*–*n*, 0) times in the result set.

Multiple **EXCEPT** operators in the same **SELECT** statement are evaluated left to right, unless parentheses dictate otherwise. **EXCEPT** binds at the same level as **UNION**.

Currently, the **FOR UPDATE**, **FOR NO KEY UPDATE**, **FOR SHARE**, and **FOR KEY SHARE** clauses cannot be specified for the result of **EXCEPT** or any input of **EXCEPT**.

- **MINUS clause**

  Has the same function and syntax as **EXCEPT** clause.

- **ORDER BY clause**

  Sorts data retrieved by **SELECT** in descending or ascending order. If the **ORDER BY** expression contains multiple columns:

  – If two columns are equal according to the leftmost expression, they are compared according to the next expression and so on.

  – If they are equal according to all specified expressions, they are returned in an implementation-dependent order.

  – When used with the **DISTINCT** keyword, the columns to be sorted in **ORDER BY** must be included in the columns of the result set retrieved by the SELECT statement.

  – When used with the **GROUP BY** clause, the columns to be sorted in **ORDER BY** must be included in the columns of the result set retrieved by the SELECT statement.

NOTICE

To support Chinese pinyin order, specify the **UTF-8**, **GB18030**, or **GBK** encoding mode during database initiation. The statements are as follows:

```
initdb –E UTF8 –D ../data –locale=zh_CN.UTF-8, initdb -E GB18030 -D ../data -locale=zh_CN.GB18030,
or initdb –E GBK –D ../data –locale=zh_CN.GBK.
```

- **LIMIT clause**

  Consists of two independent sub-clauses:

  LIMIT { count | ALL }

  OFFSET start count specifies the maximum number of rows to return, while **start** specifies the number of rows to skip before starting to return rows. When both are specified, **start** rows are skipped before starting to count the **count** rows to be returned.

- **OFFSET clause**

  The SQL: 2008 standard has introduced a different clause:

  OFFSET start { ROW | ROWS }

  start specifies the number of rows to skip before starting to return rows.

- **FETCH { FIRST | NEXT } [count] { ROW | ROWS } ONLY**

  If **count** is omitted in a **FETCH** clause, it defaults to **1**.

- **Locking clause**

  The **FOR UPDATE** clause locks the rows retrieved by **SELECT**. This prevents these rows from being modified or deleted by other transactions before the current transaction ends. That is, other transactions that attempt to run **UPDATE**, **DELETE**, **SELECT FOR UPDATE**, **SELECT FOR NO KEY UPDATE**, **SELECT FOR SHARE**, or **SELECT FOR KEY SHARE** for these rows will be blocked until the current transaction ends. Any **DELETE** on a row will also acquire the **FOR UPDATE** locking mode, as will **UPDATE** that modifies values on the primary key column. Conversely, **SELECT FOR UPDATE** waits for concurrent transactions that have run the preceding commands on the same row, and then locks and returns the updated row (there may be no row because the row may have been deleted).

  FOR NO KEY UPDATE behaves similarly to **FOR UPDATE**, except that it acquires a weaker lock that will not block **SELECT FOR KEY SHARE** that attempts to acquire the lock on the same row. Any **UPDATE** that does not acquire the **FOR UPDATE** lock will also acquire this locking mode.

  FOR SHARE behaves similarly, except that it acquires a shared rather than exclusive lock on each retrieved row. A shared lock blocks other transactions from executing **UPDATE**, **DELETE**, **SELECT FOR UPDATE**, or **SELECT FOR NO KEY UPDATE**, but does not block **SELECT FOR SHARE** or **SELECT FOR KEY SHARE**.

  FOR KEY SHARE is similar to **FOR SHARE** except that its lock is weak. **SELECT FOR UPDATE** is blocked but **SELECT FOR NO KEY UPDATE** is not blocked. A key-shared lock blocks other transactions from executing **DELETE** or **UPDATE** that modifies the key value, but does not block **UPDATE**, **SELECT FOR NO KEY UPDATE**, **SELECT FOR SHARE**, or **SELECT FOR KEY SHARE**.

  To prevent the operation from waiting for the commit of other transactions, you can use **NOWAIT**. If the selected row cannot be locked immediately, an error is reported immediately and there is no waiting. If you use **WAIT N** and the selected row cannot be locked immediately, the operation needs to wait for *N* seconds (the value of *N* is of the int type with a range of 0 ≤ *N* ≤ 2147483). If the lock is obtained within *N* seconds, the operation is performed normally. Otherwise, an error is reported.

  If specified tables are named in a locking clause, then only rows coming from those tables are locked; any other tables used in **SELECT** are simply read as usual. Otherwise, locking all tables in the statement.

  If a locking clause is applied to a view or sub-query, it affects all tables used in the view or sub-query.

  Multiple locking clauses can be written if it is necessary to specify different locking behaviors for different tables.

  If a table appears (or implicitly appears) in multiple clauses at the same time, the strongest lock is used. Similarly, a table is processed as **NOWAIT** if that is specified in any of the clauses affecting it.

  NOTICE

  FOR UPDATE, **NO KEY UPDATE**, **SHARE**, and **KEY SHARE** cannot be used to query column-store tables.

  Only **FOR SHARE** and **FOR UPDATE** can be used to query the Ustore table.

- **NLS_SORT**

  Specifies that a field is sorted in a special order. Currently, only Chinese Pinyin and case-insensitive sorting are supported. To support this sorting mode, you need to set the encoding format to UTF8, GB18030, or GBK when creating a database. If you set the encoding format to another format, for example, SQL_ASCII, an error may be reported or the sorting mode may be invalid.

  Value range:

  – **SCHINESE_PINYIN_M**, sorted by Pinyin order.

  – **generic_m_ci**: sorted in case-insensitive order (optional; only English characters are supported in the case-insensitive order.)

- **PARTITION clause**

  Queries data in the specified partition in a partitioned table.

Examples

```
-- Obtain the temp_t temporary table by a subquery and query all records in this table.
openGauss=# WITH temp_t(name,isdba) AS (SELECT usename,usesuper FROM pg_user) SELECT * FROM
temp_t;

-- Query all r_reason_sk records in the tpcds.reason table and delete duplicate records.
openGauss=# SELECT DISTINCT(r_reason_sk) FROM tpcds.reason;

-- Example of a LIMIT clause: Obtain a record from the table.
openGauss=# SELECT * FROM tpcds.reason LIMIT 1;

-- Query all records and sort them in alphabetic order.
openGauss=# SELECT r_reason_desc FROM tpcds.reason ORDER BY r_reason_desc;

-- Use table aliases to obtain data from the pg_user and pg_user_status tables:
openGauss=# SELECT a.usename,b.locktime FROM pg_user a,pg_user_status b WHERE a.usesysid=b.roloid;

-- Example of the FULL JOIN clause: Join data in the pg_user and pg_user_status tables.
openGauss=# SELECT a.usename,b.locktime,a.usesuper FROM pg_user a FULL JOIN pg_user_status b on
a.usesysid=b.roloid;

-- Example of the GROUP BY clause: Filter data based on query conditions, and group the results.
openGauss=# SELECT r_reason_id, AVG(r_reason_sk) FROM tpcds.reason GROUP BY r_reason_id HAVING
AVG(r_reason_sk) > 25;

-- Example of the GROUP BY CUBE clause: Filter data based on query conditions, and group the results.
openGauss=# SELECT r_reason_id,AVG(r_reason_sk) FROM tpcds.reason GROUP BY
CUBE(r_reason_id,r_reason_sk);

-- Example of the GROUP BY GROUPING SETS clause: Filter data based on query conditions, and group the
results.
openGauss=# SELECT r_reason_id,AVG(r_reason_sk) FROM tpcds.reason GROUP BY GROUPING
SETS((r_reason_id,r_reason_sk),r_reason_sk);

-- Example of the UNION clause: Merge the names started with W and N in the r_reason_desc column in
the tpcds.reason table.
openGauss=# SELECT r_reason_sk, tpcds.reason.r_reason_desc
    FROM tpcds.reason
    WHERE tpcds.reason.r_reason_desc LIKE 'W%'
UNION
SELECT r_reason_sk, tpcds.reason.r_reason_desc
    FROM tpcds.reason
    WHERE tpcds.reason.r_reason_desc LIKE 'N%';

-- Example of the NLS_SORT clause: Sort by Chinese Pinyin.
openGauss=# SELECT * FROM tpcds.reason ORDER BY NLSSORT( r_reason_desc, 'NLS_SORT =
SCHINESE_PINYIN_M');
```

```
-- sorting in case-insensitive order (optional; only English characters are supported in the case-insensitive
order.)
openGauss=# SELECT * FROM tpcds.reason ORDER BY NLSSORT( r_reason_desc, 'NLS_SORT =
generic_m_ci');

-- Create a range-partitioned table tpcds.reason_p.
openGauss=# CREATE TABLE tpcds.reason_p
(
  r_reason_sk integer,
  r_reason_id character(16),
  r_reason_desc character(100)
)
PARTITION BY RANGE (r_reason_sk)
(
  partition P_05_BEFORE values less than (05),
  partition P_15 values less than (15),
  partition P_25 values less than (25),
  partition P_35 values less than (35),
  partition P_45_AFTER values less than (MAXVALUE)
)
;

-- Insert data.
openGauss=# INSERT INTO tpcds.reason_p values(3,'AAAAAAAABAAAAAAA','reason 1'),
(10,'AAAAAAAABAAAAAAA','reason 2'),(4,'AAAAAAAABAAAAAAA','reason 3'),
(10,'AAAAAAAABAAAAAAA','reason 4'),(10,'AAAAAAAABAAAAAAA','reason 5'),
(20,'AAAAAAAACAAAAAAA','reason 6'),(30,'AAAAAAAACAAAAAAA','reason 7');

-- Example of the PARTITION clause: Obtain data from the P_05_BEFORE partition in the tpcds.reason_p
table.
openGauss=#  SELECT * FROM tpcds.reason_p PARTITION (P_05_BEFORE);
 r_reason_sk |   r_reason_id    |   r_reason_desc
-------------+------------------+------------------------------------
           4 | AAAAAAAABAAAAAAA | reason 3
           3 | AAAAAAAABAAAAAAA | reason 1
(2 rows)

-- Example of the GROUP BY clause: Group records in the tpcds.reason_p table by r_reason_id, and count
the number of records in each group.
openGauss=# SELECT COUNT(*),r_reason_id FROM tpcds.reason_p GROUP BY r_reason_id;
 count |   r_reason_id
-------+------------------
     2 | AAAAAAAACAAAAAAA
     5 | AAAAAAAABAAAAAAA
(2 rows)

-- Example of the GROUP BY CUBE clause: Filter data based on query conditions, and group the results.
openGauss=# SELECT * FROM tpcds.reason GROUP BY  CUBE (r_reason_id,r_reason_sk,r_reason_desc);

-- Example of the GROUP BY GROUPING SETS clause: Filter data based on query conditions, and group the
results.
openGauss=# SELECT * FROM tpcds.reason GROUP BY  GROUPING SETS
((r_reason_id,r_reason_sk),r_reason_desc);

-- Example of the HAVING clause: Group records in the tpcds.reason_p table by r_reason_id, count the
number of records in each group, and display only values whose number of r_reason_id is greater than 2.
openGauss=# SELECT COUNT(*) c,r_reason_id FROM tpcds.reason_p GROUP BY r_reason_id HAVING c>2;
 c |   r_reason_id
---+------------------
 5 | AAAAAAAABAAAAAAA
(1 row)

-- Example of the IN clause: Group records in the tpcds.reason_p table by r_reason_id, count the number
of records in each group, and display only the numbers of records whose r_reason_id is
AAAAAAAABAAAAAAA or AAAAAAAADAAAAAAA.
openGauss=# SELECT COUNT(*),r_reason_id FROM tpcds.reason_p GROUP BY r_reason_id HAVING
r_reason_id IN('AAAAAAAABAAAAAAA','AAAAAAAADAAAAAAA');
count |   r_reason_id
-------+------------------
```

```
    5 | AAAAAAAABAAAAAAA
(1 row)

-- Example of the INTERSECT clause: Query records whose r_reason_id is AAAAAAAABAAAAAAA and
whose r_reason_sk is smaller than 5.
openGauss=# SELECT * FROM tpcds.reason_p WHERE r_reason_id='AAAAAAAABAAAAAAA' INTERSECT
SELECT * FROM tpcds.reason_p WHERE r_reason_sk<5;
 r_reason_sk |   r_reason_id    |      r_reason_desc
-------------+------------------+------------------------------------
           4 | AAAAAAAABAAAAAAA | reason 3
           3 | AAAAAAAABAAAAAAA | reason 1
(2 rows)

-- Example of the EXCEPT clause: Query records whose r_reason_id is AAAAAAAABAAAAAAA and whose
r_reason_sk is greater than or equal to 4.
openGauss=# SELECT * FROM tpcds.reason_p WHERE r_reason_id='AAAAAAAABAAAAAAA' EXCEPT SELECT *
FROM tpcds.reason_p WHERE r_reason_sk<4;
r_reason_sk |   r_reason_id    |      r_reason_desc
-------------+------------------+------------------------------------
          10 | AAAAAAAABAAAAAAA | reason 2
          10 | AAAAAAAABAAAAAAA | reason 5
          10 | AAAAAAAABAAAAAAA | reason 4
           4 | AAAAAAAABAAAAAAA | reason 3
(4 rows)

-- Specify the operator (+) in the WHERE clause to indicate a left join.
openGauss=# select t1.sr_item_sk ,t2.c_customer_id from store_returns t1, customer t2 where
t1.sr_customer_sk  = t2.c_customer_sk(+)
order by 1 desc limit 1;
 sr_item_sk | c_customer_id
------------+---------------
      18000 |
(1 row)

-- Specify the operator (+) in the WHERE clause to indicate a right join.
openGauss=# select t1.sr_item_sk ,t2.c_customer_id from store_returns t1, customer t2 where
t1.sr_customer_sk(+)  = t2.c_customer_sk
order by 1 desc limit 1;
 sr_item_sk |  c_customer_id
------------+------------------
            | AAAAAAAAJNGEBAAA
(1 row)

-- Specify the operator (+) in the WHERE clause to indicate a left join and add a join condition.
openGauss=# select t1.sr_item_sk ,t2.c_customer_id from store_returns t1, customer t2 where
t1.sr_customer_sk  = t2.c_customer_sk(+) and t2.c_customer_sk(+) < 1 order by 1  limit 1;
 sr_item_sk | c_customer_id
------------+---------------
          1 |
(1 row)

-- If the operator (+) is specified in the WHERE clause, do not use expressions connected through AND/OR.
openGauss=# select t1.sr_item_sk ,t2.c_customer_id from store_returns t1, customer t2 where
not(t1.sr_customer_sk  = t2.c_customer_sk(+) and t2.c_customer_sk(+) < 1);
ERROR:  Operator "(+)" can not be used in nesting expression.
LINE 1: ...tomer_id from store_returns t1, customer t2 where not(t1.sr_...
                                                              ^
-- If the operator (+) is specified in the WHERE clause which does not support expression macros, an error
will be reported.
openGauss=# select t1.sr_item_sk ,t2.c_customer_id from store_returns t1, customer t2 where
(t1.sr_customer_sk  = t2.c_customer_sk(+))::bool;
ERROR:  Operator "(+)" can only be used in common expression.

-- If the operator (+) is specified on both sides of an expression in the WHERE clause, an error will be
reported.
openGauss=# select t1.sr_item_sk ,t2.c_customer_id from store_returns t1, customer t2 where
t1.sr_customer_sk(+)  = t2.c_customer_sk(+);
ERROR:  Operator "(+)" can't be specified on more than one relation in one join condition
HINT:  "t1", "t2"...are specified Operator "(+)" in one condition.
```

```
-- Delete the table.
openGauss=# DROP TABLE tpcds.reason_p;

-- Example of a flashback query
-- Create the tpcds.time_table table.
openGauss=#  create table tpcds.time_table(idx integer, snaptime timestamp, snapcsn bigint, timeDesc
character(100));
-- Insert records into the tpcds.time_table table.
openGauss=#  INSERT INTO tpcds.time_table select 1, now(),int8in(xidout(next_csn)), 'time1' from
gs_get_next_xid_csn();
openGauss=#  INSERT INTO tpcds.time_table select 2, now(),int8in(xidout(next_csn)), 'time2' from
gs_get_next_xid_csn();
openGauss=#  INSERT INTO tpcds.time_table select 3, now(),int8in(xidout(next_csn)), 'time3' from
gs_get_next_xid_csn();
openGauss=#  INSERT INTO tpcds.time_table select 4, now(),int8in(xidout(next_csn)), 'time4' from
gs_get_next_xid_csn();
openGauss=#  select * from tpcds.time_table;

 idx |          snaptime          | snapcsn |                                       timedesc
-----+----------------------------+---------
+------------------------------------------------------------------------------------------------------
   1 | 2021-04-25 17:50:05.360326 |  107322 | time1
   2 | 2021-04-25 17:50:10.886848 |  107324 | time2
   3 | 2021-04-25 17:50:16.12921  |  107327 | time3
   4 | 2021-04-25 17:50:22.311176 |  107330 | time4
(4 rows)
openGauss=#  delete tpcds.time_table;
DELETE 4
openGauss=#  SELECT * FROM tpcds.time_table TIMECAPSULE TIMESTAMP to_timestamp('2021-04-25
17:50:22.311176','YYYY-MM-DD HH24:MI:SS.FF');
 idx |          snaptime          | snapcsn |                                       timedesc
-----+----------------------------+---------
+------------------------------------------------------------------------------------------------------
   1 | 2021-04-25 17:50:05.360326 |  107322 | time1
   2 | 2021-04-25 17:50:10.886848 |  107324 | time2
   3 | 2021-04-25 17:50:16.12921  |  107327 | time3
(3 rows)
openGauss=#  SELECT * FROM tpcds.time_table TIMECAPSULE CSN 107330;
 idx |          snaptime          | snapcsn |                                       timedesc
-----+----------------------------+---------
+------------------------------------------------------------------------------------------------------
   1 | 2021-04-25 17:50:05.360326 |  107322 | time1
   2 | 2021-04-25 17:50:10.886848 |  107324 | time2
   3 | 2021-04-25 17:50:16.12921  |  107327 | time3
(3 rows)

-- Example of a WITH RECURSIVE query: Calculate the accumulated value from 1 to 100.
openGauss=#  WITH RECURSIVE t1(a) as (
    select 100
),
t(n) AS (
    VALUES (1)
  UNION ALL
    SELECT n+1 FROM t WHERE n < (select max(a) from t1)
)
SELECT sum(n) FROM t;
 sum
------
 5050
(1 row)
```

11.14.171 SELECT INTO

Function

SELECT INTO defines a new table based on a query result and inserts data obtained by query to the new table.

Different from **SELECT**, data found by **SELECT INTO** is not returned to the client. The table columns have the same names and data types as the output columns of the **SELECT**.

Precautions

CREATE TABLE AS provides functions similar to **SELECT INTO** in functions and provides a superset of functions provided by **SELECT INTO**. You are advised to use **CREATE TABLE AS**, because **SELECT INTO** cannot be used in a stored procedure.

Syntax

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
SELECT [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]
    { * | {expression [ [ AS ] output_name ]} [, ...] }
    INTO [ [ GLOBAL | LOCAL ] [ TEMPORARY | TEMP ] | UNLOGGED ] [ TABLE ] new_table
    [ FROM from_item [, ...] ]
    [ WHERE condition ]
    [ GROUP BY expression [, ...] ]
    [ HAVING condition [, ...] ]
    [ WINDOW {window_name AS ( window_definition )} [, ...] ]
    [ { UNION | INTERSECT | EXCEPT | MINUS } [ ALL | DISTINCT ] select ]
    [ ORDER BY {expression [ [ ASC | DESC | USING operator ] | nlssort_expression_clause ] [ NULLS { FIRST |
LAST } ]} [, ...] ]
    [ LIMIT { count | ALL } ]
    [ OFFSET start [ ROW | ROWS ] ]
    [ FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY ]
    [ {FOR { UPDATE | SHARE } [ OF table_name [, ...] ] [ NOWAIT |WAIT N]} [...] ];
```

Parameter Description

- **new_table**

  new_table specifies the name of the new table.

- **UNLOGGED**

  Specifies that the table is created as an unlogged table. Data written to unlogged tables is not written to the WALs, which makes them considerably faster than ordinary tables. However, they are not crash-safe: an unlogged table is automatically truncated after a crash or unclean shutdown. Contents of an unlogged table are also not replicated to standby servers. Any indexes created on an unlogged table are automatically unlogged as well.

  – Usage scenario: Unlogged tables do not ensure data security. Users can back up data before using unlogged tables; for example, users should back up the data before a system upgrade.

  – Troubleshooting: If data is missing in the indexes of unlogged tables due to some unexpected operations such as an unclean shutdown, users should re-create the indexes with errors.

- **GLOBAL | LOCAL**

  When creating a temporary table, you can specify the **GLOBAL** or **LOCAL** keyword before **TEMP** or **TEMPORARY**. If the keyword **GLOBAL** is specified, GaussDB creates a global temporary table. Otherwise, GaussDB creates a local temporary table.

- **TEMPORARY | TEMP**

  If **TEMP** or **TEMPORARY** is specified, the created table is a temporary table. Temporary tables are classified into global temporary tables and local temporary tables. If the keyword **GLOBAL** is specified when a temporary table is created, the table is a global temporary table. Otherwise, the table is a local temporary table.

The metadata of the global temporary table is visible to all sessions. After the sessions end, the metadata still exists. The user data, indexes, and statistics of a session are isolated from those of another session. Each session can only view and modify the data submitted by itself. Global temporary tables have two schemas: **ON COMMIT PRESERVE ROWS** and **ON COMMIT PRESERVE ROWS**. In session-based **ON COMMIT PRESERVE ROWS** schema, user data is automatically cleared when a session ends. In transaction-based **ON COMMIT DELETE ROWS** schema, user data is automatically cleared when the commit or rollback operation is performed. If the **ON COMMIT** option is not specified during table creation, the session level is used by default. Different from local temporary tables, you can specify a schema that does not start with **pg_temp_** when creating a global temporary table.

A local temporary table is automatically dropped at the end of the current session. Therefore, you can create and use temporary tables in the current session as long as the connected database node in the session is normal. Temporary tables are created only in the current session. If a DDL statement involves operations on temporary tables, a DDL error will be generated. Therefore, you are not advised to perform operations on temporary tables in DDL statements. **TEMP** is equivalent to **TEMPORARY**.

NOTICE

- Local temporary tables are visible to the current session through the schema starting with **pg_temp**. Users should not delete schemas starting with **pg_temp** or **pg_toast_temp**.
- If **TEMPORARY** or **TEMP** is not specified when you create a table but its schema is set to that starting with **pg_temp_** in the current session, the table will be created as a temporary table.
- If global temporary tables and indexes are being used by other sessions, do not perform **ALTER** or **DROP**.
- The DDL of a global temporary table affects only the user data and indexes of the current session. For example, **TRUNCATE**, **REINDEX**, and **ANALYZE** are valid only for the current session.

NOTE

For details about other **SELECT INTO** parameters, see **Parameter Description** in SELECT.

Examples

```
-- Add the values that are less than 5 in the r_reason_sk field in the tpcds.reason table to the new table.
openGauss=# SELECT * INTO tpcds.reason_t1 FROM tpcds.reason WHERE r_reason_sk < 5;
INSERT 0 6

-- Delete the tpcds.reason_t1 table.
openGauss=# DROP TABLE tpcds.reason_t1;
```

Helpful Links

SELECT

Suggestions

- **DATABASE**

  You are not advised to re-index a database in a transaction.
- **SYSTEM**

  You are not advised to re-index system catalogs in transactions.

11.14.172 SET

Function

SET modifies a run-time parameter.

Precautions

Most run-time parameters can be modified by executing **SET**. Some parameters cannot be modified after a server or session starts.

Syntax

- Set the system time zone.
  ```
  SET [ SESSION | LOCAL ] TIME ZONE { timezone | LOCAL | DEFAULT };
  ```
- Set the schema of the table.
  ```
  SET [ SESSION | LOCAL ]
      {CURRENT_SCHEMA { TO | = } { schema | DEFAULT }
      | SCHEMA 'schema'};
  ```
- Set client encoding.
  ```
  SET [ SESSION | LOCAL ] NAMES encoding_name;
  ```
- Set XML parsing mode.
  ```
  SET [ SESSION | LOCAL ] XML OPTION { DOCUMENT | CONTENT };
  ```
- Set other run-time parameters.
  ```
  SET [ LOCAL | SESSION ]
      { {config_parameter { { TO | = } { value | DEFAULT }
                          | FROM CURRENT }}};
  ```

Parameter Description

- **SESSION**

  Specifies that the specified parameters take effect for the current session. This is the default value if neither **SESSION** nor **LOCAL** appears.

  If **SET** or **SET SESSION** is executed within a transaction that is later aborted, the effects of the **SET** statement disappear when the transaction is rolled back. Once the surrounding transaction is committed, the effects will persist until the end of the session, unless overridden by another **SET**.
- **LOCAL**

  Specifies that the specified parameters take effect for the current transaction. After **COMMIT** or **ROLLBACK**, the session-level setting takes effect again.

  The effects of **SET LOCAL** last only till the end of the current transaction, whether committed or not. A special case is **SET** followed by **SET LOCAL** within a single transaction: the **SET LOCAL** value will be seen until the end of the transaction, but afterward (if the transaction is committed) the **SET** value will take effect.

- **TIME ZONE timezone**

  Specifies the local time zone for the current session.

  Value range: a valid local time zone. The corresponding run-time parameter is **TimeZone**. The default value is **PRC**.

- **CURRENT_SCHEMA**

  schema

  Specifies the current schema.

  Value range: an existing schema name If the schema name does not exist, the value of **CURRENT_SCHEMA** will be empty.

- **SCHEMA schema**

  Specifies the current schema. Here the schema is a string.

  Example: set schema 'public';

- **NAMES encoding_name**

  Specifies the client character encoding. This statement is equivalent to **set client_encoding to encoding_name**.

  Value range: a valid character encoding name. The run-time parameter corresponding to this option is **client_encoding**. The default encoding is **UTF8**.

- **XML OPTION option**

  Specifies the XML parsing mode.

  Value range: **CONTENT** (default) and **DOCUMENT**

- **config_parameter**

  Specifies the name of a configurable run-time parameter. You can use **SHOW ALL** to view available run-time parameters.

  NOTE

  Some parameters that viewed by **SHOW ALL** cannot be set by **SET**. For example, **max_datanodes**.

- **value**

  Specifies the new value of **config_parameter**. This parameter can be specified as string constants, identifiers, numbers, or comma-separated lists of these. **DEFAULT** can be written to indicate resetting the parameter to its default value.

Examples

```
-- Set the search path of a schema.
openGauss=# SET search_path TO tpcds, public;

-- Set the date style to the traditional POSTGRES style (date placed before month).
openGauss=# SET datestyle TO postgres,dmy;
```

Helpful Links

RESET and **SHOW**

11.14.173 SET CONSTRAINTS

Function

SET CONSTRAINTS sets the behavior of constraint checking within the current transaction.

IMMEDIATE constraints are checked at the end of each statement. **DEFERRED** constraints are not checked until transaction commit. Each constraint has its own **IMMEDIATE** or **DEFERRED** mode.

Upon creation, a constraint is given one of three characteristics **DEFERRABLE INITIALLY DEFERRED**, **DEFERRABLE INITIALLY IMMEDIATE**, or **NOT DEFERRABLE**. The third class is always **IMMEDIATE** and is not affected by the **SET CONSTRAINTS** statement. The first two classes start every transaction in specified modes, but its behaviors can be changed within a transaction by **SET CONSTRAINTS**.

SET CONSTRAINTS with a list of constraint names changes the mode of just those constraints (which must all be deferrable). If multiple constraints match a name, the name is affected by all of these constraints. **SET CONSTRAINTS ALL** changes the modes of all deferrable constraints.

When **SET CONSTRAINTS** changes the mode of a constraint from **DEFERRED** to **IMMEDIATE**, the new mode takes effect retroactively: any outstanding data modifications that would have been checked at the end of the transaction are instead checked during the execution of the **SET CONSTRAINTS** statement. If any such constraint is violated, the **SET CONSTRAINTS** fails (and does not change the constraint mode). Therefore, **SET CONSTRAINTS** can be used to force checking of constraints to occur at a specific point in a transaction.

Constraints are always checked immediately when a row is inserted or modified.

Precautions

SET CONSTRAINTS sets the behavior of constraint checking only within the current transaction. Therefore, if you execute this statement outside of a transaction block (**START TRANSACTION/COMMIT** pair), it will not appear to have any effect.

Syntax

```
SET CONSTRAINTS  { ALL  |  { name  }  [, ...]  }  { DEFERRED  | IMMEDIATE  } ;
```

Parameter Description

- **name**

  Specifies the constraint name.

  Value range: an existing table name, which can be found in the system catalog **pg_constraint**.

- **ALL**

  Specifies all constraints.

- **DEFERRED**

  Specifies that constraints are not checked until transaction commit.

- **IMMEDIATE**

  Specifies that constraints are checked at the end of each statement.

Examples

```
-- Set that constraints are checked when a transaction is committed.
openGauss=# SET CONSTRAINTS ALL DEFERRED;
```

11.14.174 SET ROLE

Function

Sets the current user identifier of the current session.

Precautions

- Users of the current session must be members of specified **rolename**, but the system administrator can choose any roles.
- Executing this statement may add rights of a user or restrict rights of a user. If the role of a session user has the **INHERITS** attribute, it automatically has all rights of roles that **SET ROLE** enables the role to be. In this case, **SET ROLE** physically deletes all rights directly granted to session users and rights of its belonging roles and only leaves rights of the specified roles. If the role of the session user has the **NOINHERITS** attribute, **SET ROLE** deletes rights directly granted to the session user and obtains rights of the specified role.

Syntax

- Set the current user identifier of the current session.
  ```
  SET [ SESSION | LOCAL ] ROLE role_name PASSWORD 'password';
  ```
- Reset the current user identifier to that of the current session.
  ```
  RESET ROLE;
  ```

Parameter Description

- **SESSION**

  Specifies that the statement takes effect only for the current session. This parameter is used by default.
- **LOCAL**

  Specifies that the specified statement takes effect only for the current transaction.
- **role_name**

  Specifies the role name.

  Value range: a string. It must comply with the identifier naming convention.
- **password**

  Specifies the password of a role. It must comply with the password convention.
- **RESET ROLE**

  Resets the current user identifier.

Examples

```
-- Create a role paul.
openGauss=# CREATE ROLE paul IDENTIFIED BY 'xxxxxxxxx';

-- Set the current user to paul.
openGauss=# SET ROLE paul PASSWORD 'xxxxxxxxx';

-- View the current session user and the current user.
openGauss=# SELECT SESSION_USER, CURRENT_USER;

-- Reset the current user.
openGauss=# RESET role;

-- Delete the user.
openGauss=# DROP USER paul;
```

11.14.175 SET SESSION AUTHORIZATION

Function

Sets the session user identifier and the current user identifier of the current SQL session to a specified user.

Precautions

The session identifier can be changed only when the initial session user has the system administrator rights. Otherwise, the system supports the statement only when the authenticated user name is specified.

Syntax

- Set the session user identifier and the current user identifier of the current session.
  ```
  SET [ SESSION | LOCAL ] SESSION AUTHORIZATION role_name PASSWORD 'password';
  ```
- Reset the identifiers of the session and current users to the initially authenticated user names.
  ```
  {SET [ SESSION | LOCAL ] SESSION AUTHORIZATION DEFAULT
     | RESET SESSION AUTHORIZATION};
  ```

Parameter Description

- **SESSION**

  Specifies that the specified parameters take effect for the current session.
- **LOCAL**

  Specifies that the specified statement takes effect only for the current transaction.
- **role_name**

  Username

  Value range: a string. It must comply with the identifier naming convention.
- **password**

  Specifies the password of a role. It must comply with the password convention.
- **DEFAULT**

Resets the identifiers of the session and current users to the initially authenticated user names.

Examples

```
-- Create a role paul.
openGauss=# CREATE ROLE paul IDENTIFIED BY 'xxxxxxxxx';

-- Set the current user to paul.
openGauss=# SET SESSION AUTHORIZATION paul password 'xxxxxxxxx';

-- View the current session user and the current user.
openGauss=# SELECT SESSION_USER, CURRENT_USER;

-- Reset the current user.
openGauss=# RESET SESSION AUTHORIZATION;

-- Delete the user.
openGauss=# DROP USER paul;
```

Reference

SET ROLE

11.14.176 SET TRANSACTION

Function

SET TRANSACTION sets characteristics of a transaction. Available transaction characteristics include the transaction isolation level and transaction access mode (read/write or read only). You can set the current transaction characteristics using **LOCAL** or the default transaction characteristics of a session using **SESSION**.

Precautions

The current transaction characteristics must be set in a transaction, that is, **START TRANSACTION** or **BEGIN** must be executed before **SET TRANSACTION** is executed. Otherwise, the setting does not take effect.

Syntax

Set the isolation level and access mode of the transaction.

```
{ SET [ LOCAL ] TRANSACTION|SET SESSION CHARACTERISTICS AS TRANSACTION }
  { ISOLATION LEVEL { READ COMMITTED | SERIALIZABLE | REPEATABLE READ }
  | { READ WRITE | READ ONLY } } [, ...]
```

Parameter Description

- **LOCAL**

  Specifies that the specified statement takes effect only for the current transaction.
- **SESSION**

  Specifies that the specified parameters take effect for the current session.

  Value range: a string. It must comply with the naming convention.
- **ISOLATION_LEVEL**

Specifies the transaction isolation level that determines the data that a transaction can view if other concurrent transactions exist.

NOTE

The isolation level cannot be changed after data is modified using **SELECT**, **INSERT**, **DELETE**, **UPDATE**, **FETCH**, or **COPY** in the current transaction.

Value range:

- **READ COMMITTED**: Only submitted data is read. It is the default value.
- **REPEATABLE READ**: Only the data committed before transaction start is read. Uncommitted data or data committed in other concurrent transactions cannot be read.
- **SERIALIZABLE**: Currently, this isolation level is not supported in GaussDB. It is equivalent to **REPEATABLE READ**.

- **READ WRITE | READ ONLY**

  Specifies the transaction access mode (read/write or read only).

Examples

```
-- Start a transaction and set its isolation level to READ COMMITTED and access mode to READ ONLY.
openGauss=# START TRANSACTION;
openGauss=# SET LOCAL TRANSACTION ISOLATION LEVEL READ COMMITTED READ ONLY;
openGauss=# COMMIT;
```

11.14.177 SHOW

Function

SHOW shows the current value of a run-time parameter.

Precautions

None

Syntax

```
SHOW
  {
    [VARIABLES LIKE] configuration_parameter |
    CURRENT_SCHEMA |
    TIME ZONE |
    TRANSACTION ISOLATION LEVEL |
    SESSION AUTHORIZATION |
    ALL
  };
```

Parameter Description

See **Parameter Description** in **RESET**.

Examples

```
-- Show the value of timezone.
openGauss=# SHOW timezone;

-- Show all parameters.
openGauss=# SHOW ALL;
```

```
-- Show all parameters whose names contain var.
openGauss=# SHOW VARIABLES LIKE var;
```

Helpful Links

SET and **RESET**

11.14.178 SHUTDOWN

Function

SHUTDOWN shuts down the currently connected database node.

Precautions

Only the administrator can run this command.

Syntax

```
SHUTDOWN
 {
        |
   fast  |
   immediate
 };
```

Parameter Description

- **""**

  If the shutdown mode is not specified, the default mode **fast** is used.
- **fast**

  Rolls back all active transactions, forcibly disconnects the client, and shuts down the database node without waiting for the client to disconnect.
- **immediate**

  Shuts down the server forcibly. Fault recovery will occur on the next startup.

Examples

```
-- Shut down the current database node.
openGauss=# SHUTDOWN;

-- Shut down the current database node in fast mode.
openGauss=# SHUTDOWN FAST;
```

11.14.179 SNAPSHOT

Function

SNAPSHOT controls data in a unified manner for multiple users.

Precautions

- The GUC parameter **db4ai_snapshot_mode** classifies the snapshot storage model into MSS and CSS. The GUC parameter

db4ai_snapshot_version_delimiter specifies the version separator and its default value is **@**. The GUC parameter **db4ai_snapshot_version_separator** specifies the sub-version separator and its default value is **.**.

- When the incremental storage mode is used for snapshots, the snapshots are dependent on each other. Snapshots must be deleted in the dependency sequence.
- The snapshot feature is used to maintain data between team members, involving data transcription between administrators and common users. Therefore, the snapshot feature is not supported in private user and separation of duty (**enableSeparationOfDuty** is set to **ON**) scenarios.
- When you need a stable and available snapshot for tasks such as AI training, you need to publish the snapshot.

Syntax

1. Create a snapshot.

   You can run the **CREATE SNAPSHOT... AS** and **CREATE SNAPSHOT... FROM** statements to create a data table snapshot.

   – CREATE SNAPSHOT AS
   ```
   CREATE SNAPSHOT <qualified_name> [@ <version | ident | sconst>]
       [COMMENT IS <sconst>}
       AS query;
   ```
   – CREATE SNAPSHOT FROM
   ```
   CREATE SNAPSHOT <qualified_name> [@ <version | ident | sconst>]
       FROM @ <version | ident | sconst>
       [COMMENT IS <sconst>}
           USING (
           { INSERT [INTO SNAPSHOT] …
             | UPDATE [SNAPSHOT] [AS <alias>] SET … [FROM …] [WHERE …]
             | DELETE [FROM SNAPSHOT] [AS <alias>] [USING …] [WHERE …]
             | ALTER [SNAPSHOT] { ADD … | DROP … } [, …]
         } [; …]
       );
   ```

2. Delete a snapshot.

   PURGE SNAPSHOT
   ```
   PURGE SNAPSHOT <qualified_name> @ <version | ident | sconst>;
   ```

3. Sample snapshots.

   SAMPLE SNAPSHOT
   ```
   SAMPLE SNAPSHOT <qualified_name> @ <version | ident | sconst>
       [STRATIFY BY attr_list]
       { AS <label> AT RATIO <num> [COMMENT IS <comment>] } [, …]
   ```

4. Publish snapshots.

   PUBLISH SNAPSHOT
   ```
   PUBLISH SNAPSHOT <qualified_name> @ <version | ident | sconst>;
   ```

5. Archive snapshots.

   ARCHIVE SNAPSHOT
   ```
   ARCHIVE SNAPSHOT <qualified_name> @ <version | ident | sconst>;
   ```

Parameter Description

- qualified_name

  Name of the snapshot to be created

  Value range: a string. It must comply with the identifier naming convention.

- version

  (Optional) Version number of a snapshot. This parameter is optional. The system automatically extends the sequence number.

  Value range: string, consisting of numbers and separators.

Examples

```
create snapshot s1@1.0 comment is 'first version' as select * from t1;
create snapshot s1@3.0 from @1.0 comment is 'inherits from @1.0' using (INSERT VALUES(6, 'john'), (7,
'tim'); DELETE WHERE id = 1);
SELECT * FROM s1@1.0;
purge snapshot s1@1.0;
sample snapshot s1@2.0 stratify by name as nick at ratio .5;
publish snapshot s1@2.0;
archive snapshot s1@2.0;
```

Helpful Links

None

11.14.180 START TRANSACTION

Function

START TRANSACTION starts a transaction. If the isolation level or read/write mode is specified, a new transaction will have those characteristics. You can also specify them using **SET TRANSACTION**.

Precautions

None

Syntax

Format 1: START TRANSACTION

```
START TRANSACTION
  [
    {
       ISOLATION LEVEL { READ COMMITTED | SERIALIZABLE | REPEATABLE READ }
       | { READ WRITE | READ ONLY }
     } [, ...]
  ];
```

Format 2: BEGIN

```
BEGIN [ WORK | TRANSACTION ]
  [
    {
       ISOLATION LEVEL { READ COMMITTED | SERIALIZABLE | REPEATABLE READ }
       | { READ WRITE | READ ONLY }
      } [, ...]
  ];
```

Parameter Description

- **WORK | TRANSACTION**

  Specifies the optional keyword in BEGIN format without functions.

- **ISOLATION LEVEL**

  Specifies the transaction isolation level that determines the data that a transaction can view if other concurrent transactions exist.

  NOTE

  The isolation level of a transaction cannot be reset after the first clause (**SELECT**, **INSERT**, **DELETE**, **UPDATE**, **FETCH**, **COPY**) for modifying data is executed in the transaction.

  Value range:

  – **READ COMMITTED**: Only submitted data is read. It is the default value.
  – **REPEATABLE READ**: Only the data committed before transaction start is read. Uncommitted data or data committed in other concurrent transactions cannot be read.
  – **SERIALIZABLE**: Currently, this isolation level is not supported in GaussDB. It is equivalent to **REPEATABLE READ**.

- **READ WRITE | READ ONLY**

  Specifies the transaction access mode (read/write or read only).

Examples

```
-- Start a transaction in default mode.
openGauss=# START TRANSACTION;
openGauss=# SELECT * FROM tpcds.reason;
openGauss=# END;

-- Start a transaction in default mode.
openGauss=# BEGIN;
openGauss=# SELECT * FROM tpcds.reason;
openGauss=# END;

-- Start a transaction with the isolation level being READ COMMITTED and the access mode being READ
WRITE:
openGauss=# START TRANSACTION ISOLATION LEVEL READ COMMITTED READ WRITE;
openGauss=# SELECT * FROM tpcds.reason;
openGauss=# COMMIT;
```

Helpful Links

COMMIT | END, **ROLLBACK**, and **SET TRANSACTION**

11.14.181 TIMECAPSULE TABLE

Function

The **TIMECAPSULE TABLE** statement restores a table to an earlier state in the event of human or application errors.

The table can flash back to a past point in time, depending on the old version of the data stored in the system. In addition, GaussDB cannot restore a table to an earlier state through DDL operations that has changed the structure of the table.

Precautions

- The **TIMECAPSULE TABLE** statement can be used to flash back the data of the old version or the data from the recycle bin.

  - **TO TIMECAPSULE** and **TO CSN** can flash back a table to an earlier version.
  - The recycle bin records the objects dropped or truncated by running **DROP** and **TRUNCATE**. **TO BEFORE DROP** and **TO BEFORE TRUNCATE** flash back from the recycle bin.
- The following object types do not support flashback: system catalogs, column-store tables, MOTs, DFS tables, global temporary tables, local temporary tables, unlogged tables, sequence tables, hash bucket tables, and encrypted tables.
- Between the flashback point and the current point, a statement (**DDL**, **DCL**, or **VACUUM FULL**) that modifies the table structure or affects physical storage has been executed. Therefore, the flashback fails.
- To run **DROP**, you must have the CREATE or USAGE permission on the schema to which the junk object belongs, and you must be the owner of the schema or the owner of the junk object.

  To run **TRUNCATE**, you must have the CREATE or USAGE permission on the schema to which the junk object belongs, and you must be the owner of the schema or the junk object. In addition, you must have the TRUNCATE permission on the junk object.
- Scenarios or tables that do not support DROP or TRUNCATE FLASHBACK
  - Scenario where the recycle bin is disabled (**enable_recyclebin** is set to **off**)
  - Scenario where the system is being maintained (**xc_maintenance_mode** is set to **on**) or is being upgraded
  - Scenario where multiple objects are deleted (The **DROP** or **TRUNCATE TABLE** command is executed to delete multiple objects at the same time.)
  - System catalogs, column-store tables, MOTs, DFS tables, global temporary tables, local temporary tables, unlogged tables, sequence tables, and hash bucket tables

Syntax

```
TIMECAPSULE TABLE [schema.]table_name TO { CSN expr | TIMESTAMP expr | BEFORE { DROP [RENAME
TO table_name] | TRUNCATE } }
```

Parameter Description

- **schema_name**

  Specifies a schema containing the table to be flashed back. If this parameter is not specified, the current schema is used.
- **table_name**

  Specifies a table name.
- **TO CSN**

  Specifies the CSN corresponding to the time point when the table is to be flashed back. *expr* must be a number representing a valid CSN.
- **TO TIMESTAMP**

  Specifies a timestamp value corresponding to the point in time to which you want to flash back the table. The result of *expr* must be a valid past timestamp (convert a string to a time type using the **TO_TIMESTAMP** function). The table will be flashed back to a time within approximately 3 seconds of the specified timestamp.

Note: When the flashback point is too old, the old version cannot be obtained because it is recycled. As a result, the flashback fails and the error message "Restore point too old" is displayed.

- **TO BEFORE DROP**

  Retrieves dropped tables and their subobjects from the recycle bin.

  You can specify either the original user-specified name of the table or the system-generated name assigned to the object when it was deleted.

  - System-generated recycle bin object names are unique. Therefore, if you specify the system-generated name, the database retrieves that specified object. To see the content in your recycle bin, run **select * from gs_recyclebin;**.
  - If you specify the user-specified name and the recycle bin contains more than one object of that name, the database retrieves the object that was moved to the recycle bin most recently. If you want to retrieve an older version of the table, then do one of these things:
    - Specify the system-generated recycle bin name of the table you want to retrieve.
    - Run the **TIMECAPSULE TABLE ... TO BEFORE DROP** statement until you retrieve the table you want.
  - When a dropped table is restored, only the base table name is restored, and the names of other subobjects remain the same as those in the recycle bin. You can run the DDL command to manually change the names of subobjects as required.
  - The recycle bin does not support write operations such as DML, DCL, and DDL, and does not support DQL query operations (supported in later versions).
  - The **recyclebin_retention_time** parameter has been set for specifying the retention period of objects in the recycle bin. The objects will be automatically deleted after the retention period expires.

- **RENAME TO**

  Specifies a new name for the table retrieved from the recycle bin.

- **TRUNCATE**

  Flashes back to the point in time before the TRUNCATE operation.

Examples

```
-- Delete the tpcds.reason_t2 table.
DROP TABLE IF EXISTS tpcds.reason_t2;
-- Create the tpcds.reason_t2 table.
openGauss=# CREATE TABLE tpcds.reason_t2
 (
  r_reason_sk    integer,
  r_reason_id    character(16),
  r_reason_desc  character(100)
  )with(storage_type = ustore);
-- Insert records into the tpcds.reason_t2 table.
openGauss=#  INSERT INTO tpcds.reason_t2 VALUES (1, 'AA', 'reason1'),(2, 'AB', 'reason2'),(3, 'AC',
'reason3');
```

```
INSERT 0 3
-- Delete data from the tpcds.reason_t2 table.
openGauss=#  TRUNCATE TABLE tpcds.reason_t2;
-- Query data in the tpcds.reason_t2 table.
openGauss=#  select * from tpcds.reason_t2;
 r_reason_sk | r_reason_id | r_reason_desc
-------------+-------------+---------------
(0 rows)
-- Perform the TRUNCATE FLASHBACK operation.
openGauss=#  TIMECAPSULE TABLE tpcds.reason_t2 to BEFORE TRUNCATE;
openGauss=#  select * from tpcds.reason_t2;
 r_reason_sk |   r_reason_id    |                                  r_reason_desc
-------------+------------------
+----------------------------------------------------------------------------------------------------
           1 | AA               | reason1
           2 | AB               | reason2
           3 | AC               | reason3
(3 rows)
-- Delete the tpcds.reason_t2 table.
openGauss=#  DROP TABLE tpcds.reason_t2;
-- Perform the DROP FLASHBACK operation.
openGauss=#  TIMECAPSULE TABLE tpcds.reason_t2 to BEFORE DROP;
TimeCapsule Table
```

11.14.182 TRUNCATE

Function

TRUNCATE quickly removes all rows from a database table.

It has the same effect as an unqualified **DELETE** on each table, but it is faster since it does not actually scan the tables. This is most useful on large tables.

Precautions

- **TRUNCATE TABLE** has the same function as a **DELETE** statement with no **WHERE** clause, emptying a table.
- **TRUNCATE TABLE** uses less system and transaction log resources as compared with **DELETE**.
  - **DELETE** deletes a row each time, and records the deletion of each row in the transaction log.
  - **TRUNCATE TABLE** deletes all rows in a table by releasing the data page storing the table data, and records the releasing of the data page only in the transaction log.
- The differences between **TRUNCATE**, **DELETE**, and **DROP** are as follows:
  - **TRUNCATE TABLE** deletes content, releases space, but does not delete definitions.
  - **DELETE TABLE** deletes content, but does not delete definitions nor release space.
  - **DROP TABLE** deletes content and definitions, and releases space.

Syntax

- Delete data from a table.

```
TRUNCATE [ TABLE ] [ ONLY ] {table_name [ * ]} [, ... ]
    [ CONTINUE IDENTITY ] [ CASCADE | RESTRICT] [ PURGE ]};
```

- Truncate the data in a partition.

```
ALTER TABLE [ IF EXISTS  ] { [ ONLY  ] table_name
                        | table_name *
                        | ONLY ( table_name )  }
   TRUNCATE PARTITION { partition_name
                    | FOR (  partition_value  [, ...] )  } [ UPDATE GLOBAL INDEX ];
```

Parameter Description

- **ONLY**

  If **ONLY** is specified, only the specified table is cleared. Otherwise, the table and all its subtables (if any) are cleared.
- **table_name**

  Specifies the name (optionally schema-qualified) of the target table.

  Value range: an existing table name
- **CONTINUE IDENTITY**

  Does not change the values of sequences. This is the default action.
- **CASCADE | RESTRICT**
  - **CASCADE**: Clears all tables that are added to a group.
  - **RESTRICT** (default value): Clears all data.
- **PURGE**: Purges table data in the recycle bin by default.
- **partition_name**

  Specifies the partition in the target partitioned table.

  Value range: an existing table name
- **partition_value**

  Specifies the value of the specified partition key.

  The value specified by **PARTITION FOR** can uniquely identify a partition.

  Value range: value range of the partition key for the partition to be renamed

  NOTICE

  When the **PARTITION FOR** clause is used, the entire partition where **partition_value** is located is cleared.
- **UPDATE GLOBAL INDEX**

  If this parameter is used, all global indexes in the partitioned table are updated to ensure that data can be queried correctly using global indexes. If this parameter is not used, all global indexes in the partitioned table will become invalid.

Examples

```
-- Create a table.
openGauss=# CREATE TABLE tpcds.reason_t1 AS TABLE tpcds.reason;

-- Truncate the tpcds.reason_t1 table.
openGauss=# TRUNCATE TABLE tpcds.reason_t1;

-- Delete the table.
openGauss=# DROP TABLE tpcds.reason_t1;
```

```
-- Create a partitioned table.
openGauss=# CREATE TABLE tpcds.reason_p
(
  r_reason_sk integer,
  r_reason_id character(16),
  r_reason_desc character(100)
)PARTITION BY RANGE (r_reason_sk)
(
  partition p_05_before values less than (05),
  partition p_15 values less than (15),
  partition p_25 values less than (25),
  partition p_35 values less than (35),
  partition p_45_after values less than (MAXVALUE)
);

-- Insert data.
openGauss=# INSERT INTO tpcds.reason_p SELECT * FROM tpcds.reason;

-- Clear the p_05_before partition.
openGauss=# ALTER TABLE tpcds.reason_p TRUNCATE PARTITION p_05_before;

-- Clear the p_15 partition.
openGauss=# ALTER TABLE tpcds.reason_p TRUNCATE PARTITION for (13);

-- Clear the partitioned table.
openGauss=# TRUNCATE TABLE tpcds.reason_p;

-- Delete the table.
openGauss=# DROP TABLE tpcds.reason_p;
```

11.14.183 UPDATE

Function

UPDATE updates data in a table. **UPDATE** changes the values of the specified columns in all rows that satisfy the condition. The **WHERE** clause clarifies conditions. The columns to be modified need to be mentioned in the **SET** clause; columns not explicitly modified retain their previous values.

Precautions

- The owner of a table, users granted with the UPDATE permission on the table, or users granted with the UPDATE ANY TABLE permission can update data in the table. The system administrator has the permission to update data in the table by default.
- You must have the SELECT permission on all tables involved in the expressions or conditions.
- For column-store tables, the RETURNING clause is currently not supported.
- Column-store tables do not support non-deterministic update. If you update data in one row with multiple rows of data in a column-store table, an error will be reported.
- Memory space that records update operations in column-store tables is not recycled. You need to clean it by executing **VACUUM FULL table_name**.
- Currently, **UPDATE** cannot be used in column-store replication tables.
- The generated column cannot be directly written. In the **UPDATE** statement, values cannot be specified for generated columns, but the keyword **DEFAULT** can be specified.

Syntax

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
UPDATE [/*+ plan_hint */] [ ONLY ] table_name [ partition_clause ] [ * ] [ [ AS ] alias ]
SET {column_name = { expression | DEFAULT }
    |( column_name [, ...] ) = {( { expression | DEFAULT } [, ...] ) |sub_query }}[, ...]
    [ FROM from_list] [ WHERE condition ]
    [ RETURNING {*
              | {output_expression [ [ AS ] output_name ]} [, ...] }];

where sub_query can be:
SELECT [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]
{ * | {expression [ [ AS ] output_name ]} [, ...] }
[ FROM from_item [, ...] ]
[ WHERE condition ]
[ GROUP BY grouping_element [, ...] ]
[ HAVING condition [, ...] ]
[ ORDER BY {expression [ [ ASC | DESC | USING operator ] | nlssort_expression_clause ] [ NULLS { FIRST |
LAST } ]} [, ...] ]
[ LIMIT { [offset,] count | ALL } ]
```

- The subquery **with_query** is as follows:

  ```
  with_query_name [ ( column_name [, ...] ) ]
      AS [ [ NOT ] MATERIALIZED ] ( {select | values | insert | update | delete} )
  ```

Parameter Description

- **WITH [RECURSIVE] with_query [, ...]**

  Specifies one or more subqueries that can be referenced by name in the main query, which is equivalent to a temporary table.

  If **RECURSIVE** is specified, it allows a subquery **SELECT** to reference itself by name.

  The detailed format of **with_query** is as follows: **with_query_name [(column_name [, ...])] AS [[NOT] MATERIALIZED] ({select | values | insert | update | delete})**

  – **with_query_name** specifies the name of the result set generated by a subquery. Such names can be used to access the result sets of subqueries in a query.

  – **column_name** specifies the column name displayed in the subquery result set.

  – Each subquery can be a **SELECT**, **VALUES**, **INSERT**, **UPDATE** or **DELETE** statement.

  – You can use **MATERIALIZED** or **NOT MATERIALIZED** to modify the CTE.

    ■ If **MATERIALIZED** is specified, the WITH query will be materialized, and a copy of the subquery result set is generated. The copy is directly queried at the reference point. Therefore, the WITH subquery cannot be jointly optimized with the SELECT statement trunk (for example, predicate pushdown and equivalence class transfer). In this scenario, you can use **NOT MATERIALIZED** for modification. If the WITH query can be executed as a subquery inline, the preceding optimization can be performed.

    ■ If the user does not explicitly declare the materialized attribute, comply with the following rules: If the CTE is referenced only once in the SELECT statement trunk to which it belongs and semantically supports inline execution, it will be rewritten as subquery inline execution. Otherwise, the materialized execution will be performed in CTE Scan mode.

- **plan_hint** clause

  Follows the **UPDATE** keyword in the **/*+ */** format. It is used to optimize the plan of an **UPDATE** statement block. For details, see **Hint-based Tuning**. In each statement, only the first **/*+** *plan_hint* ***/** comment block takes effect as a hint. Multiple hints can be written.

- **table_name**

  Specifies the name (optionally schema-qualified) of the table to be updated.

  Value range: an existing table name

- **partition_clause**

  Updates a specified partition.

  PARTITION { (partition_name) | FOR (partition_value [, ...]) } |

  SUBPARTITION { (subpartition_name) | FOR (subpartition_value [, ...]) }

  For details about the keywords, see **SELECT**.

  For details, see **CREATE TABLE SUBPARTITION**.

- **alias**

  Specifies a substitute name for the target table.

  Value range: a string. It must comply with the identifier naming convention.

- **column_name**

  Specifies the name of the column to be modified.

  You can refer to this column by specifying the target table alias and the column name. Example:

  UPDATE foo AS f SET f.col_name = 'namecol';

  Value range: an existing column

- **expression**

  Specifies a value assigned to a column or an expression that assigns the value.

- **DEFAULT**

  Specifies the default value of a column.

  The value is **NULL** if no specified default value has been assigned to it.

- **sub_query**

  Specifies a subquery.

  This statement can be executed to update a table with information for other tables in the same database. For details about clauses in the **SELECT** statement, see **SELECT**.

  When a single column is updated, the ORDER BY and LIMIT clauses can be used. When multiple columns are updated, the ORDER BY and LIMIT clauses cannot be used.

- **from_list**

  Specifies a list of table expressions, allowing columns from other tables to appear in the **WHERE** condition and the update expressions. This is similar to the list of tables that can be specified in the **FROM** clause of a **SELECT** statement.

NOTICE

Note that the target table must not appear in the **from_list**, unless you intend a self-join (in which case it must appear with an alias in the **from_list**).

- **condition**

  Specifies an expression that returns a value of type Boolean. Only rows for which this expression returns **true** are updated. You are not advised to use numeric types such as int for **condition**, because such types can be implicitly converted to bool values (non-zero values are implicitly converted to **true** and **0** is implicitly converted to **false**), which may cause unexpected results.
- **output_expression**

  Specifies an expression to be computed and returned by the **UPDATE** statement after each row is updated.

  Value range: The expression can use any column names of the table named by **table_name** or tables listed in **FROM**. Write * to return all columns.
- **output_name**

  Specifies a name to use for a returned column.

Examples

```
-- Create the student1 table.
openGauss=# CREATE TABLE student1
(
   stuno     int,
   classno   int
);

-- Insert data.
openGauss=# INSERT INTO student1 VALUES(1,1);
openGauss=# INSERT INTO student1 VALUES(2,2);
openGauss=# INSERT INTO student1 VALUES(3,3);

-- View data.
openGauss=# SELECT * FROM student1;

-- Update the values of all records.
openGauss=# UPDATE student1 SET classno = classno*2;

-- View data.
openGauss=# SELECT * FROM student1;

-- Delete the table.
openGauss=# DROP TABLE student1;
```

11.14.184 VACUUM

Function

VACUUM recycles storage space occupied by tables or **B-Tree** indexes. In normal database operation, rows that have been deleted are not physically removed from their table; they remain present until a **VACUUM** is done. Therefore, it is necessary to do **VACUUM** periodically, especially on frequently-updated tables.

Precautions

- With no table specified, **VACUUM** processes all the tables that the current user has permission to vacuum in the current database. With a parameter, **VACUUM** processes only that table.
- To perform VACUUM operation to a table, you must be a table owner or a user granted the VACUUM permission on the table. By default, the system administrator has this permission. However, database owners are allowed to VACUUM all tables in their databases, except shared catalogs. (The restriction for shared catalogs means that a true database-wide VACUUM can only be executed by the system administrator). **VACUUM** skips over any tables that the calling user does not have the permission to vacuum.
- **VACUUM** cannot be executed inside a transaction block.
- It is recommended that active production databases be vacuumed frequently (at least nightly), in order to remove dead rows. After adding or deleting a large number of rows, it might be a good idea to run **VACUUM ANALYZE** for the affected table. This will update the system catalogs with the results of all recent changes, and allow the query planner to make better choices in planning queries.
- **FULL** is recommended only in special scenarios. For example, you wish to physically narrow the table to decrease the occupied disk space after deleting most rows of a table. **VACUUM FULL** usually shrinks a table more than **VACUUM** does. The **FULL** option does not clear indexes. You are advised to periodically run the **REINDEX** statement. Deleting all indexes, running **VACUUM FULL**, and rebuilding indexes is usually a faster choice. If the physical space usage does not decrease after you run the statement, check whether there are other active transactions (that have started before you delete data transactions and not ended before you run **VACUUM FULL**). If there are such transactions, run this statement again when the transactions quit.
- **VACUUM** causes a substantial increase in I/O traffic, which might cause poor performance for other active sessions. Therefore, it is sometimes advisable to use the cost-based VACUUM delay feature.
- When **VERBOSE** is specified, **VACUUM** prints progress messages to indicate which table is currently being processed. Various statistics about the tables are printed as well. However, if you execute **VACUUM** and specify the **VERBOSE** option for column-store tables, no information is returned.
- When the option list is surrounded by parentheses, the options can be written in any order. If there are no brackets, the options must be given in the order displayed in the syntax.
- **VACUUM** and **VACUUM FULL** clear deleted tuples after the delay specified by **vacuum_defer_cleanup_age**.
- **VACUUM ANALYZE** executes a VACUUM operation and then an ANALYZE operation for each selected table. This is a handy combination form for routine maintenance scripts.
- Plain **VACUUM** (without **FULL**) simply recycles space and makes it available for reuse. This form of statement can operate in parallel with normal reading and writing of the table, as an exclusive lock is not obtained. **VACUUM FULL** executes wider processing, including moving rows across blocks to compress tables so they occupy the minimum number of disk blocks. This form is much

slower and requires an exclusive lock on each table while it is being processed.

- When you do **VACUUM** to a column-store table, the following operations are internally performed: data in the delta table is migrated to the primary table, and the delta and desc tables of the primary table are vacuumed. **VACUUM** does not recycle the storage space of the delta table. To recycle it, do **VACUUM DELTAMERGE** to the column-store table.
- A deadlock may occur when multiple **VACUUM FULL** statements are executed simultaneously.
- If the **xc_maintenance_mode** parameter is not enabled, the **VACUUM FULL** operation will skip all system catalogs.
- If you run **VACUUM FULL** immediately after running **DELETE**, the space will not be recycled. After executing **DELETE**, execute 1000 non-SELECT transactions, or wait for 1s and then execute one transaction. Then, run **VACUUM FULL** to the space.

Syntax

- Recycle space and update statistics information, without requirements for keyword orders.
```
VACUUM [ ( { FULL | FREEZE | VERBOSE | {ANALYZE | ANALYSE }} [,...] ) ]
    [ table_name [ (column_name [, ...] ) ] [ PARTITION ( partition_name ) | SUBPARTITION
( subpartition_name ) ] ];
```
- Recycle space, without updating statistics information.
```
VACUUM [ FULL [COMPACT] ] [ FREEZE ] [ VERBOSE ] [ table_name
[ PARTITION ( partition_name )  | SUBPARTITION ( subpartition_name ) ] ];
```
- Recycle space and update statistics information, and require keywords in order.
```
VACUUM [ FULL ] [ FREEZE ] [ VERBOSE ] { ANALYZE | ANALYSE } [ VERBOSE ]
    [ table_name [ (column_name [, ...] ) ] ] [ PARTITION ( partition_name )];
```

Parameter Description

- **FULL**

  Selects "FULL" vacuum, which can recycle more space, but takes much longer and exclusively locks the table.

  NOTE

  1. Using FULL will cause statistics missing. To collect statistics, add the keyword ANALYZE to VACUUM FULL.
  2. The Ustore engine does not support VACUUM FULL. After VACUUM FULL is executed, "INFO: skipping "test" --- Ustore table does not support vacuum full" is printed.

- **FREEZE**

  Is equivalent to running **VACUUM** with the **vacuum_freeze_min_age** parameter set to **zero**.

- **VERBOSE**

  Prints a detailed VACUUM activity report for each table.

- **ANALYZE | ANALYSE**

  Updates statistics used by the planner to determine the most efficient way to execute a query.

NOTE

VACUUM is also triggered when **autovacuum** is set to **analyze** for an Ustore partitioned table.

- **table_name**

  Specifies the name (optionally schema-qualified) of a specific table to vacuum.

  Value range: name of a specific table to vacuum. Defaults are all tables in the current database.

- **column_name**

  Specifies the name of the column to be analyzed. This parameter must be used together with **ANALYZE**.

  Value range: name of the column to be analyzed Defaults are all columns.

- **PARTITION**

  COMPACT and **PARTITION** cannot be used at the same time.

- **partition_name**

  Specifies the level-1 partition name of the table to be cleared. If it is left empty, all level-1 partitions are cleared.

- **subpartition_name**

  Specifies the level-2 partition name of the table to be cleared. If it is left empty, all level-2 partitions are cleared.

- **DELTAMERGE**

  (For column-store tables) Migrates data from the delta table to primary tables. For a column-store table, this operation is controlled by **deltarow_threshold**. For details, see enable_delta_store and **Parameter Description**.

Examples

```
-- Create an index in the tpcds.reason tabletable.
openGauss=# CREATE UNIQUE INDEX ds_reason_index1 ON tpcds.reason(r_reason_sk);

-- Vacuum the tpcds.reason table that has indexes.
openGauss=# VACUUM (VERBOSE, ANALYZE) tpcds.reason;

-- Delete the index.
openGauss=# DROP INDEX ds_reason_index1 CASCADE;
openGauss=# DROP TABLE tpcds.reason;
```

Suggestions

- vacuum
  - **VACUUM** cannot be executed inside a transaction block.
  - It is recommended that active production databases be vacuumed frequently (at least nightly), in order to remove dead rows. It is strongly recommended that you run **VACUUM ANALYZE** after adding or deleting a large number of records.
  - **FULL** is recommended only in special scenarios. For example, you wish to physically narrow the table to decrease the occupied disk space after deleting most rows of a table.

– Before performing the VACUUM FULL operation, you are advised to delete all indexes in related tables, run **VACUUM FULL**, and then re-create the index.

11.14.185 VALUES

Function

VALUES computes a row or a set of rows based on given values. It is most commonly used to generate a constant table within a large statement.

Precautions

- **VALUES** lists with large numbers of rows should be avoided, as you might encounter out-of-memory failures or poor performance. **VALUES** appearing within **INSERT** is a special case, because the desired column types are known from the **INSERT**'s target table, and need not be inferred by scanning the **VALUES** list. In this case, **VALUE** can handle larger lists than are practical in other contexts.
- If more than one row is specified, all the rows must have the same number of elements.

Syntax

```
VALUES {( expression [, ...] )} [, ...]
    [ ORDER BY { sort_expression [ ASC | DESC | USING operator ] } [, ...] ]
    [ LIMIT { count | ALL } ]
    [ OFFSET start [ ROW | ROWS ] ]
    [ FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY ];
```

Parameter Description

- **expression**

  Specifies a constant or expression to compute and insert at the indicated place in the resulting table or set of rows.

  In a **VALUES** list appearing at the top level of an **INSERT**, an expression can be replaced by **DEFAULT** to indicate that the destination column's default value should be inserted. **DEFAULT** cannot be used when **VALUES** appears in other contexts.
- **sort_expression**

  Specifies an expression or integer constant indicating how to sort the result rows.
- **ASC**

  Specifies an ascending sort order.
- **DESC**

  Specifies a descending sort order.
- **operator**

  Specifies a sorting operator.
- **count**

  Specifies the maximum number of rows to return.

- **OFFSET start { ROW | ROWS }**

  Specifies the maximum number of returned rows, whereas **start** specifies the number of rows to skip before starting to return rows.
- **FETCH { FIRST | NEXT } [count] { ROW | ROWS } ONLY**

  The **FETCH** clause restricts the total number of rows starting from the first row of the return query result, and the default value of **count** is **1**.

Examples

See **Examples** in **INSERT**.

11.15 Appendix

11.15.1 GIN Indexes

11.15.1.1 Introduction

Generalized Inverted Index (GIN) is designed for handling cases where the items to be indexed are composite values, and the queries to be handled by the index need to search for element values that appear within the composite items. For example, the items could be documents, and the queries could be searches for documents containing specific words.

We use the word "item" to refer to a composite value that is to be indexed, and the word "key" to refer to an element value. GIN stores and searches for keys, not item values.

A GIN index stores a set of (key, posting list) key-value pairs, where a posting list is a set of row IDs in which the key occurs. The same row ID can appear in multiple posting lists, since an item can contain more than one key. Each key value is stored only once, so a GIN index is very compact for cases where the same key appears many times.

GIN is generalized in the sense that the GIN access method code does not need to know the specific operations that it accelerates. Instead, it uses custom strategies defined for particular data types. The strategy defines how keys are extracted from indexed items and query conditions, and how to determine whether a row that contains some of the key values in a query actually satisfies the query.

11.15.1.2 Scalability

The GIN interface has a high level of abstraction, requiring the access method implementer only to implement the semantics of the data type being accessed. The GIN layer itself takes care of concurrency, logging and searching the tree structure.

All it takes to get a GIN access method working is to implement multiple user-defined methods, which define the behavior of keys in the tree and the relationships between keys, indexed items, and indexable queries. In short, GIN combines extensibility with generality, code reuse, and a clean interface.

There are four methods that an operator class for GIN must provide:

- int compare(Datum a, Datum b)

  Compares two keys (not indexed items) and returns an integer less than zero, zero, or greater than zero, indicating whether the first key is less than, equal to, or greater than the second. Null keys are never passed to this function.

- Datum *extractValue(Datum itemValue, int32 *nkeys, bool **nullFlags)

  Returns an array of keys given an item to be indexed. The number of returned keys must be stored into ***nkeys**. If any of the keys can be null, also palloc an array of ***nkeys** bool fields, store its address at ***nullFlags**, and set these null flags as needed. ***nullFlags** can be left **NULL** (its initial value) if all keys are non-null. The returned value can be **NULL** if the item contains no keys.

- Datum *extractQuery(Datum query, int32 *nkeys, StrategyNumber n, bool **pmatch, Pointer **extra_data, bool **nullFlags, int32 *searchMode)

  Returns a palloc'd array of keys given a value to be queried; that is, query is the value on the right-hand side of an indexable operator whose left-hand side is the indexed column. n is the strategy number of the operator within the operator class. Often, **extractQuery** will need to consult n to determine the data type of query and the method it should use to extract key values. The number of returned keys must be stored into ***nkeys**. If any of the keys can be null, also palloc an array of ***nkeys** bool fields, store its address at ***nullFlags**, and set these null flags as needed. ***nullFlags** can be left NULL (its initial value) if all keys are non-null. The returned value can be NULL if the query contains no keys.

  searchMode is an output argument that allows **extractQuery** to specify details about how the search will be done. If ***searchMode** is set to **GIN_SEARCH_MODE_DEFAULT** (which is the value it is initialized to before call), only items that match at least one of the returned keys are considered candidate matches. If ***searchMode** is set to **GIN_SEARCH_MODE_INCLUDE_EMPTY**, then in addition to items containing at least one matching key, items that contain no keys at all are considered candidate matches. (This mode is useful for implementing is-subset-of operators, for example.) If ***searchMode** is set to **GIN_SEARCH_MODE_ALL**, then all non-null items in the index are considered candidate matches, whether they match any of the returned keys or not.

  pmatch is an output argument for use when partial match is supported. To use it, **extractQuery** must allocate an array of *nkeys Booleans and store its address at ***pmatch**. Each element of the array should be set to **TRUE** if the corresponding key requires partial match, **FALSE** if not. If ***pmatch** is set to **NULL** then GIN assumes partial match is not required. The variable is initialized to NULL before call, so this argument can simply be ignored by operator classes that do not support partial match.

  extra_data is an output argument that allows **extractQuery** to pass additional data to the **consistent** and **comparePartial** methods. To use it, **extractQuery** must allocate an array of ***nkeys** pointers and store its address at ***extra_data**, then store whatever it wants to into the individual pointers. The variable is initialized to **NULL** before call, so this argument can simply be ignored by operator classes that do not require extra data. If ***extra_data** is set, the whole array is passed to the **consistent** method, and the appropriate element to the **comparePartial** method.

- bool consistent(bool check[], StrategyNumber n, Datum query, int32 nkeys, Pointer extra_data[], bool *recheck, Datum queryKeys[], bool nullFlags[])

  Returns **TRUE** if an indexed item satisfies the query operator with StrategyNumber **n** (or might satisfy it, if the recheck indication is returned). This function does not have direct access to the indexed item's value, since GIN does not store items explicitly. Rather, what is available is knowledge about which key values extracted from the query appear in a given indexed item. The check array has length **nkeys**, which is the same as the number of keys previously returned by **extractQuery** for this query datum. Each element of the check array is **TRUE** if the indexed item contains the corresponding query key, for example, if (check[i] == TRUE), the i-th key of the **extractQuery** result array is present in the indexed item. The original query datum is passed in case the **consistent** method needs to consult it, and so are the **queryKeys[]** and **nullFlags[]** arrays previously returned by **extractQuery**. **extra_data** is the extra-data array returned by **extractQuery**, or **NULL** if none.

  When **extractQuery** returns a null key in **queryKeys[]**, the corresponding **check[]** element is **TRUE** if the indexed item contains a null key; that is, the semantics of **check[]** are like **IS NOT DISTINCT FROM**. The **consistent** function can examine the corresponding **nullFlags[]** element if it needs to tell the difference between a regular value match and a null match.

  On success, ***recheck** should be set to **TRUE** if the heap tuple needs to be rechecked against the query operator, or **FALSE** if the index test is exact. That is, a **FALSE** return value guarantees that the heap tuple does not match the query; a **TRUE** return value with ***recheck** set to **FALSE** guarantees that the heap tuple matches the query; and a **TRUE** return value with ***recheck** set to **TRUE** means that the heap tuple might match the query, so it needs to be fetched and rechecked by evaluating the query operator directly against the originally indexed item.

Optionally, an operator class for GIN can supply the following method:

- int comparePartial(Datum partial_key, Datum key, StrategyNumber n, Pointer extra_data)

  Compares a partial-match query key to an index key. Returns an integer whose sign indicates the result: less than zero means the index key does not match the query, but the index scan should continue; zero means that the index key matches the query; greater than zero indicates that the index scan should stop because no more matches are possible. The strategy number **n** of the operator that generated the partial match query is provided, in case its semantics are needed to determine when to end the scan. Also, **extra_data** is the corresponding element of the extra-data array made by **extractQuery**, or **NULL** if none. Null keys are never passed to this function.

To support "partial match" queries, an operator class must provide the **comparePartial** method, and its **extractQuery** method must set the **pmatch** parameter when a partial-match query is encountered. For details, see **Partial Match Algorithm**.

The actual data types of the various Datum values mentioned in this section vary depending on the operator class. The item values passed to **extractValue** are always of the operator class's input type, and all key values must be of the class's **STORAGE** type. The type of the query argument passed to **extractQuery**, consistent and **triConsistent** is whatever is specified as the right-hand input type of the class member operator identified by the strategy number. This need not be the same as the item type, so long as key values of the correct type can be extracted from it.

11.15.1.3 Implementation

Internally, a GIN index contains a B-tree index constructed over keys, where each key is an element of one or more indexed items (a member of an array, for example) and where each tuple in a page contains either a pointer to a B-tree of heap pointers (a "posting tree"), or a simple list of heap pointers (a "posting list") when the list is small enough to fit into a single index tuple along with the key value.

Multi-column GIN indexes are implemented by building a single B-tree over composite values (column number, key value). The key values for different columns can be of different types.

GIN Fast Update Technique

Updating a GIN index tends to be slow because of the intrinsic nature of inverted indexes: inserting or updating one heap row can cause many inserts into the index. After the table is vacuumed or if the pending list becomes larger than **work_mem**, the entries are moved to the main GIN data structure using the same bulk insert techniques used during initial index creation. This greatly increases the GIN index update speed, even counting the additional vacuum overhead. Moreover the overhead work can be done by a background process instead of in foreground query processing.

The main disadvantage of this approach is that searches must scan the list of pending entries in addition to searching the regular index, and so a large list of pending entries will slow searches significantly. Another disadvantage is that, while most updates are fast, an update that causes the pending list to become "too large" will incur an immediate cleanup cycle and be much slower than other updates. Proper use of autovacuum can minimize both of these problems.

If consistent response time (of entity cleanup and of update) is more important than update speed, use of pending entries can be disabled by turning off the **fastupdate** storage parameter for a GIN index. For details, see **CREATE INDEX**.

Partial Match Algorithm

GIN can support "partial match" queries, in which the query does not determine an exact match for one or more keys, but the possible matches fall within a narrow range of key values (within the key sorting order determined by the **compare** support method). The **extractQuery** method, instead of returning a key value to be matched exactly, returns a key value that is the lower bound of the range to be searched, and sets the **pmatch** flag true. The key range is then scanned using the **comparePartial** method. **comparePartial** must return zero for a matching index key, less than zero for a non-match that is still within the range to be searched, or greater than zero if the index key is past the range that could match.

11.15.1.4 GIN Tips and Tricks

Create vs. Insert

Insertion into a GIN index can be slow due to the likelihood of many keys being inserted for each item. So, for bulk insertions into a table it is advisable to drop the GIN index and recreate it after finishing bulk insertion. GUC parameters related to GIN index creation and query performance as follows:

- maintenance_work_mem

  Build time for a GIN index is very sensitive to the **maintenance_work_mem** setting.

- work_mem

  During a series of insertions into an existing GIN index that has **FASTUPDATE** enabled, the system will clean up the pending-entry list whenever the list grows larger than **work_mem**. To avoid fluctuations in observed response time, it is desirable to have pending-list cleanup occur in the background (that is, via autovacuum). Foreground cleanup operations can be avoided by increasing **work_mem** or making **autovacuum** more aggressive. However, increasing **work_mem** means that if a foreground cleanup occurs, it will take even longer.

- gin_fuzzy_search_limit

  The primary goal of developing GIN indexes was to support highly scalable full-text search in GaussDB. A full-text search often returns a very large set of results. This often happens when the query contains very frequent words, so that the large result set is not even useful. Since reading many tuples from the disk and sorting them could take a lot of time, this is unacceptable for production. To facilitate controlled execution of such queries, GIN has a configurable soft upper limit on the number of rows returned: the **gin_fuzzy_search_limit** configuration parameter. The default value **0** indicates that there is no limit on the returned set. If a non-zero limit is set, then the returned set is a subset of the whole result set, chosen at random. **Soft upper limit** means that the actual number of returned results may deviate from the specified limit, depending on the quality of the query and the system random number generator.

11.15.2 Extended Functions

The following table lists the extended functions supported by GaussDB. These functions are for reference only.

Category	Name	Description
Access privilege inquiry function	has_sequence_privilege(user, sequence, privilege)	Queries whether a specified user has privilege for sequences.
	has_sequence_privilege(sequence, privilege)	Queries whether the current user has privilege for sequence.
Trigger function	pg_get_triggerdef(oid)	Gets **CREATE** [**CONSTRAINT**] **TRIGGER** command for triggers.
	pg_get_triggerdef(oid, boolean)	Gets **CREATE** [**CONSTRAINT**] **TRIGGER** command for triggers.

11.15.3 Extended Syntax

GaussDB provides the following extended syntax:

Table 11-124 Extended SQL syntax

Category	Keywords	Description
Creating a table	**INHERITS (parent_table [, ...])**	Specifies whether an inherited table is supported.
	column_constraint: **REFERENCES reftable [(refcolumn)] [MATCH FULL | MATCH PARTIAL | MATCH SIMPLE][ON DELETE action] [ON UPDATE action]**	You can run **REFERENCES reftable[(***refcolumn***)] [MATCH FULL |MATCH PARTIAL | MATCH SIMPLE] [ON DELETE action] [ON UPDATE action]** to create foreign key constraints for tables.
Loading a module	**CREATE EXTENSION**	Loads a new module to the current database.
	DROP EXTENSION	Deletes the loaded module.
Aggregate functions	**CREATE AGGREGATE**	Defines a new aggregate function.
	ALTER AGGREGATE	Modifies the definition of an aggregate function.
	DROP AGGREGATE	Drops an existing function.

12 Stored Procedure

12.1 Stored Procedure

In GaussDB, business rules and logics are saved as stored procedures.

A stored procedure is a combination of SQL and PL/SQL. Stored procedures can move the code that executes business rules from applications to databases. Therefore, the code storage can be used by multiple programs at a time.

For details about how to create and call a stored procedure, see **CREATE PROCEDURE**.

12.2 Data Types

A data type refers to a value set and an operation set defined on the value set. The GaussDBdatabase consists of tables, each of which is defined by its own columns. Each column corresponds to a data type. GaussDB uses corresponding functions to perform operations on data based on data types. For example, GaussDB can perform addition, subtraction, multiplication, and division operations on data of numeric values.

12.3 Data Type Conversion

Certain data types in the database support implicit data type conversions, such as assignments and parameters called by functions. For other data types, you can use the type conversion functions provided by GaussDB, such as the **CAST** function, to forcibly convert them.

GaussDB lists common implicit data type conversions in **Table 12-1**.

NOTICE

The valid value range of **DATE** supported by GaussDB is from 4713 B.C. to 294276 A.D.

Table 12-1 Implicit data type conversions

Raw Data Type	Target Data Type	Remarks
CHAR	VARCHAR2	-
CHAR	NUMBER	Raw data must consist of digits.
CHAR	DATE	Raw data cannot exceed the valid date range.
CHAR	RAW	-
CHAR	CLOB	-
VARCHAR2	CHAR	-
VARCHAR2	NUMBER	Raw data must consist of digits.
VARCHAR2	DATE	Raw data cannot exceed the valid date range.
VARCHAR2	CLOB	-
NUMBER	CHAR	-
NUMBER	VARCHAR2	-
DATE	CHAR	-
DATE	VARCHAR2	-
RAW	CHAR	-
RAW	VARCHAR2	-
CLOB	CHAR	-
CLOB	VARCHAR2	-
CLOB	NUMBER	Raw data must consist of digits.
INT4	CHAR	-
INT4	BOOLEAN	-
BOOLEAN	INT4	-

12.4 Arrays, Sets, and Records

12.4.1 Arrays

Use of Array Types

Before the use of arrays, an array type needs to be defined.

Define an array type immediately after the **AS** keyword in a stored procedure. The definition method is as follows:

```
TYPE array_type IS VARRAY(size) OF data_type;
```

In the preceding information:

- **array_type**: indicates the name of the array type to be defined.
- **VARRAY**: indicates the array type to be defined.
- **size**: indicates the maximum number of members in the array to be defined. The value is a positive integer.
- **data_type**: indicates the types of members in the array to be created.

NOTE

- In GaussDB, an array automatically increases. If an access violation occurs, a null value is returned, and no error message is reported.
- The scope of an array type defined in a stored procedure takes effect only in this stored procedure.
- It is recommended that you use one of the preceding methods to define an array type. If both methods are used to define the same array type, GaussDB prefers the array type defined in a stored procedure to declare array variables.
- **data_type** can also be the record type defined in a stored procedure (anonymous blocks are not supported), but cannot be the array or set type defined in the stored procedure.

GaussDB supports access to array elements by using parentheses, and it also supports the **extend**, **count**, **first**, **last**, **prior**, **exists**, **trim**, **next**, and **delete** functions.

NOTE

- If a stored procedure contains a DML statement (such as **SELECT**, **UPDATE**, **INSERT**, or **DELETE**), you are advised to use square brackets to access array elements. Using parentheses will access arrays by default. If no array exists, function expressions will be identified.
- Exercise caution when using the **DELETE** statement to delete a single element. Otherwise, the element sequence may be incorrect.
- When the CLOB size is greater than 1 GB, the table of type, record type, and CLOB cannot be used in the input or output parameter, cursor, or raise info in a stored procedure.

Examples

```
-- Perform operations on an array in the stored procedure.
openGauss=# CREATE OR REPLACE PROCEDURE array_proc AS
DECLARE
      TYPE ARRAY_INTEGER IS VARRAY(1024) OF INTEGER;-- Define the array type.
      ARRINT ARRAY_INTEGER: = ARRAY_INTEGER();  -- Declare the variable of the array type.
BEGIN
      ARRINT.extend(10);
      FOR I IN 1..10 LOOP
             ARRINT(I) := I;
      END LOOP;
```

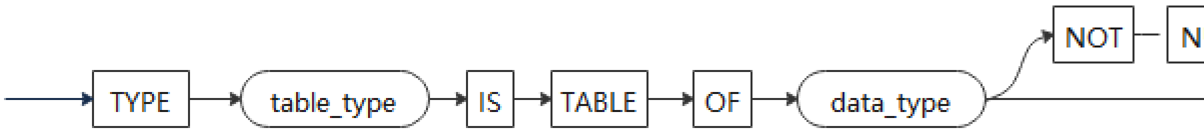
```
    DBE_OUTPUT.PRINT_LINE(ARRINT.COUNT);
    DBE_OUTPUT.PRINT_LINE(ARRINT(1));
    DBE_OUTPUT.PRINT_LINE(ARRINT(10));
    DBE_OUTPUT.PRINT_LINE(ARRINT(ARRINT.FIRST));
    DBE_OUTPUT.PRINT_LINE(ARRINT(ARRINT.LAST));
    DBE_OUTPUT.PRINT_LINE(ARRINT(ARRINT.NEXT(ARRINT.FIRST)));
    DBE_OUTPUT.PRINT_LINE(ARRINT(ARRINT.PRIOR(ARRINT.LAST)));
    ARRINT.TRIM();

    IF ARRINT.EXISTS(10) THEN
        DBE_OUTPUT.PRINT_LINE('Exist 10th element');
    ELSE
        DBE_OUTPUT.PRINT_LINE('Not exist 10th element');
    END IF;
    DBE_OUTPUT.PRINT_LINE(ARRINT.COUNT);
    DBE_OUTPUT.PRINT_LINE(ARRINT(ARRINT.FIRST));
    DBE_OUTPUT.PRINT_LINE(ARRINT(ARRINT.LAST));
    ARRINT.DELETE();
END;
/

-- Invoke the stored procedure.
openGauss=# CALL array_proc();

-- Delete the stored procedure.
openGauss=# DROP PROCEDURE array_proc;
```

12.4.2 Sets

Use of Set Types

Before the use of sets, a set type needs to be defined.

Define a set type immediately after the **AS** keyword in a stored procedure. The definition method is as follows:



In the preceding information:

- **table_type**: indicates the name of the set type to be defined.
- **TABLE**: indicates the set type to be defined.
- **data_type**: indicates the types of members in the set to be created.
- **indexby_type**: indicates the type of the set index to be created.

NOTE

- In GaussDB, a set automatically increases. If an access violation occurs, a null value is returned, and no error message is reported.
- The scope of a set type defined in a stored procedure takes effect only in this stored procedure.
- The index can only be of the integer or varchar type. The length of the varchar type is not restricted.
- **NOT NULL** has no function but only takes effect in the syntax.
- **data_type** can also be the record type or set type defined in a stored procedure (anonymous blocks are not supported), but cannot be the array type.
- Variables of the nested set type cannot be used across packages.
- Variables of the **TABLE OF index by** type cannot be nested in a record as the input and output parameters of a stored procedure.
- Variables of the **TABLE OF index by** type cannot be used as input and output parameters of functions.
- The **RAISE INFO** command cannot be used to print the entire nested **TABLE OF** variable.
- The **TABLE OF** variable cannot be transferred across autonomous transactions.
- The input and output parameters of a stored procedure cannot be defined as the nested **TABLE OF** type.

GaussDB supports access to set elements by using parentheses, and it also supports the **extend**, **count**, **first**, **last**, **prior**, **next**, and **delete** functions.

The set functions support **multiset union**, **intersect**, **except all**, and **distinct**.

NOTE

- An expression can contain only one variable of the **TABLE OF index by** type.
- Exercise caution when using the **DELETE** statement to delete a single element. Otherwise, the element sequence may be incorrect.

Examples

```
-- Perform operations on a set in the stored procedure.
openGauss=# CREATE OR REPLACE PROCEDURE table_proc AS
DECLARE
      TYPE TABLE_INTEGER IS TABLE OF INTEGER;-- Define the set type.
      TABLEINT TABLE_INTEGER := TABLE_INTEGER();  -- Declare the variable of the set type.
BEGIN
      TABLEINT.extend(10);
      FOR I IN 1..10 LOOP
             TABLEINT(I) := I;
      END LOOP;
      DBE_OUTPUT.PRINT_LINE(TABLEINT.COUNT);
      DBE_OUTPUT.PRINT_LINE(TABLEINT(1));
      DBE_OUTPUT.PRINT_LINE(TABLEINT(10));
END;
/

-- Invoke the stored procedure.
openGauss=# CALL table_proc();

-- Delete the stored procedure.
openGauss=# DROP PROCEDURE table_proc;

-- Perform operations on a nested table in the stored procedure.
openGauss=# CREATE OR REPLACE PROCEDURE nest_table_proc AS
DECLARE
```

```
    TYPE TABLE_INTEGER IS TABLE OF INTEGER;-- Define the set type.
    TYPE NEST_TABLE_INTEGER IS TABLE OF TABLE_INTEGER;-- Define the set type.
    NEST_TABLE_VAR NEST_TABLE_INTEGER; -- Declare a variable of the nested table type.
BEGIN
    FOR I IN 1..10 LOOP
        NEST_TABLE_VAR(I)(I) := I;
    END LOOP;
    DBE_OUTPUT.PRINT_LINE(NEST_TABLE_VAR.COUNT);
    DBE_OUTPUT.PRINT_LINE(NEST_TABLE_VAR(1)(1));
    DBE_OUTPUT.PRINT_LINE(NEST_TABLE_VAR(10)(10));
END;
/

-- Invoke the stored procedure.
openGauss=# CALL nest_table_proc();

-- Delete the stored procedure.
openGauss=# DROP PROCEDURE nest_table_proc;
```

12.4.3 record

record Variables

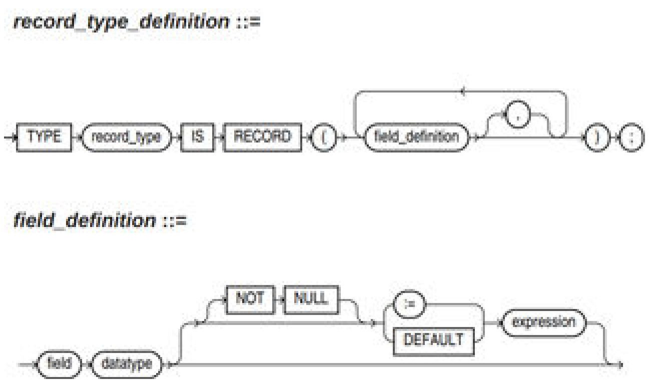
Perform the following operations to create a record variable:

Define a record type and use this type to declare a variable.

Syntax

For the syntax of the record type, see **Figure 12-1**.

Figure 12-1 Syntax of the record type



The above syntax diagram is explained as follows:

- **record_type**: record name
- **field**: record columns
- **datatype**: record data type
- **expression**: expression for setting a default value

NOTE

In GaussDB:

- When assigning values to record variables, you can:
  - Declare a record type and define member variables of this type when you declare a function or stored procedure.
  - Assign the value of a record variable to another record variable.
  - Use **SELECT INTO** or **FETCH** to assign values to a record type.
  - Assign the **NULL** value to a record variable.
- The **INSERT** and **UPDATE** statements cannot use a record variable to insert or update data.
- Just like a variable, a record column of the compound type does not have a default value in the declaration.
- The data type can be the record type, array type, or set type defined in a stored procedure (anonymous blocks are not supported).

Examples

```
The table definition used in the following is defined as follows:
openGauss=# \d emp_rec
                Table "public.emp_rec"
  Column  |              Type              | Modifiers
----------+--------------------------------+-----------
 empno    | numeric(4,0)                   | not null
 ename    | character varying(10)          |
 job      | character varying(9)           |
 mgr      | numeric(4,0)                   |
 hiredate | timestamp(0) without time zone |
 sal      | numeric(7,2)                   |
 comm     | numeric(7,2)                   |
 deptno   | numeric(2,0)                   |

-- Perform array operations in the function.
openGauss=# CREATE OR REPLACE FUNCTION regress_record(p_w VARCHAR2)
RETURNS
VARCHAR2  AS $$
DECLARE

   -- Declare a record type.
   type rec_type is record (name  varchar2(100), epno int);
   employer rec_type;

   -- Use %type to declare the record type.
   type rec_type1 is record (name  emp_rec.ename%type, epno int not null :=10);
   employer1 rec_type1;

   -- Declare a record type with a default value.
   type rec_type2 is record (
         name varchar2 not null := 'SCOTT',
         epno int not null :=10);
    employer2 rec_type2;
    CURSOR C1 IS  select ename,empno from emp_rec order by 1 limit 1;

BEGIN
      -- Assign a value to a member record variable.
     employer.name := 'WARD';
     employer.epno = 18;
     raise info 'employer name: % , epno:%', employer.name, employer.epno;

      -- Assign the value of a record variable to another variable.
     employer1 := employer;
     raise info 'employer1 name: % , epno: %',employer1.name, employer1.epno;

      -- Assign the NULL value to a record variable.
     employer1 := NULL;
     raise info 'employer1 name: % , epno: %',employer1.name, employer1.epno;

      -- Obtain the default value of a record variable.
     raise info 'employer2 name: % ,epno: %', employer2.name, employer2.epno;

      -- Use a record variable in the FOR loop.
      for employer in select ename,empno from emp_rec order by 1  limit 1
          loop
               raise info 'employer name: % , epno: %', employer.name, employer.epno;
          end loop;

      -- Use a record variable in the SELECT INTO statement.
      select ename,empno  into employer2 from emp_rec order by 1 limit 1;
      raise info 'employer name: % , epno: %', employer2.name, employer2.epno;

      -- Use a record variable in a cursor.
      OPEN C1;
      FETCH C1 INTO employer2;
      raise info 'employer name: % , epno: %', employer2.name, employer2.epno;
      CLOSE C1;
      RETURN employer.name;
END;
$$
LANGUAGE plpgsql;

-- Call this function.
openGauss=# CALL regress_record('abc');

-- Delete the function.
openGauss=# DROP FUNCTION regress_record;
```

12.5 DECLARE Syntax

12.5.1 Basic Structure

Structure

A PL/SQL block can contain a sub-block which can be placed in any section. The following describes the architecture of a PL/SQL block:

- **DECLARE**: declares variables, types, cursors, and regional stored procedures and functions used in the PL/SQL block.
  ```
  DECLARE
  ```
  NOTE

  This part is optional if no variable needs to be declared.
  - An anonymous block may omit the **DECLARE** keyword if no variable needs to be declared.
  - For a stored procedure, **AS** is used, which is equivalent to **DECLARE**. The **AS** keyword must be reserved even if there is no variable declaration part.
- **EXECUTION**: specifies procedure and SQL statements. It is the main part of a program. Mandatory.
  ```
  BEGIN
  ```
- Exception part: processes errors. Optional.
  ```
  EXCEPTION
  ```
- End
  ```
  END;
  /
  ```

NOTICE

You are not allowed to use consecutive tabs in the PL/SQL block because they may result in an exception when the **gsql** tool is executed with the **-r** parameter specified.

Category

PL/SQL blocks are classified into the following types:

- Anonymous block: a dynamic block that can be executed only for once. For details about the syntax, see **Figure 12-2**.
- Subprogram: a stored procedure, function, operator, or packages stored in a database. A subprogram created in a database can be called by other programs.

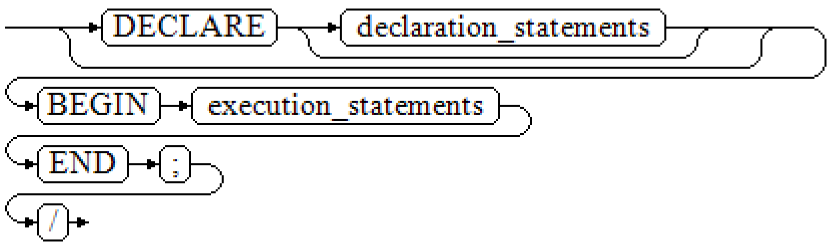
12.5.2 Anonymous Blocks

An anonymous block applies to a script infrequently executed or a one-off activity. An anonymous block is executed in a session and is not stored.

Syntax

Figure 12-2 shows the syntax diagrams for an anonymous block.

Figure 12-2 anonymous_block::=



Details about the syntax diagram are as follows:

- The execute part of an anonymous block starts with a **BEGIN** statement, has a break with an **END** statement, and ends with a semicolon (;). Type a slash (/) and press **Enter** to execute the statement.

NOTICE

The terminator "/" must be written in an independent row.

- The declaration section includes the variable definition, type, and cursor definition.
- A simplest anonymous block does not execute any commands. At least one statement, even a **NULL** statement, must be presented in any implementation blocks.

12.5.3 Subprogram

A subprogram stores stored procedures, functions, operators, and advanced packages. A subprogram created in a database can be called by other programs.

12.6 Basic Statements

During PL/SQL programming, you may define some variables, assign values to variables, and call other stored procedures. This chapter describes basic PL/SQL statements, including variable definition statements, value assignment statements, call statements, and return statements.

NOTE

You are advised not to call the SQL statements containing passwords in the stored procedures because authorized users may view the stored procedure file in the database and password information is leaked. If a stored procedure contains other sensitive information, permission to access this procedure must be configured, preventing information leakage.

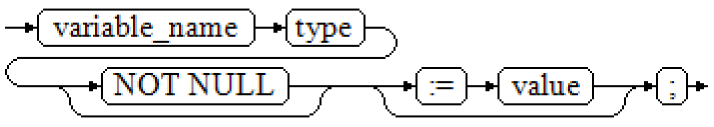
12.6.1 Variable Definition Statements

This section describes the declaration of variables in the PL/SQL and the scope of this variable in codes.

Variable Declaration

For details about the variable declaration syntax, see **Figure 12-3**.

Figure 12-3 declare_variable::=



The above syntax diagram is explained as follows:

- **variable_name** indicates the name of a variable.
- **type** indicates the type of a variable.
- **value** indicates the initial value of the variable. (If the initial value is not given, NULL is taken as the initial value.) **value** can also be an expression.

Examples

```
openGauss=# DECLARE
    emp_id  INTEGER := 7788; -- Define a variable and assign a value to it.
BEGIN
    emp_id := 5*7784; -- Assign a value to the variable.
END;
/
```

In addition to the declaration of basic variable types, **%TYPE** and **%ROWTYPE** can be used to declare variables related to table columns or table structures.

%TYPE Attribute

%TYPE declares a variable to be of the same data type as a previously declared variable (for example, a column in a table). For example, if you want to define a *my_name* variable whose data type is the same as the data type of the **firstname** column in the **employee** table, you can define the variable as follows:

```
my_name employee.firstname%TYPE
```

In this way, you can declare *my_name* without the need of knowing the data type of **firstname** in **employee**, and the data type of *my_name* can be automatically updated when the data type of **firstname** changes.

```
TYPE employee_record is record (id INTEGER, firstname VARCHAR2(20));
my_employee employee_record;
my_id my_employee.id%TYPE;
my_id_copy my_id%TYPE;
```

%ROWTYPE Attribute

%ROWTYPE declares data types of a set of data. It stores a row of table data or results fetched from a cursor. For example, if you want to define a set of data with the same column names and column data types as the **employee** table, you can define the data as follows:

```
my_employee employee%ROWTYPE
```

The attribute can also be used on the cursor. The column names and column data types of this set of data are the same as those of the **employee** table. For the cursor in a package, **%ROWTYPE** can be omitted. **%TYPE** can also reference the type of a column in the cursor. You can define the data as follows:

```
cursor cur is select * from employee;
my_employee cur%ROWTYPE
my_name cur.firstname%TYPE
my_employee2 cur -- For the cursor defined in a package, %ROWTYPE can be omitted.
```

NOTICE

- **%TYPE** cannot reference the type of a composite variable or a record variable, a column type of the record type, a column type of a variable of the cross-package composite type, or a column type of a cursor variable of the cross-package type.
- **%ROWTYPE** cannot reference the type of a composite variable or a record variable and the type of a cross-package cursor.

Scope of a Variable

The scope of a variable indicates the accessibility and availability of the variable in code block. In other words, a variable takes effect only within its scope.

- To define a function scope, a variable must declare and create a **BEGIN-END** block in the declaration section. The necessity of such declaration is also determined by block structure, which requires that a variable has different scopes and lifetime during a process.
- A variable can be defined multiple times in different scopes, and inner definition can cover outer one.

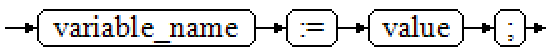
- A variable defined in an outer block can also be used in a nested block. However, the outer block cannot access variables in the nested block.

12.6.2 Assignment Statements

Syntax

Figure 12-4 shows the syntax diagram for assigning a value to a variable.

Figure 12-4 assignment_value::=



The above syntax diagram is explained as follows:

- **variable_name** indicates the name of a variable.
- **value** can be a value or an expression. The type of **value** must be compatible with the type of **variable_name**.

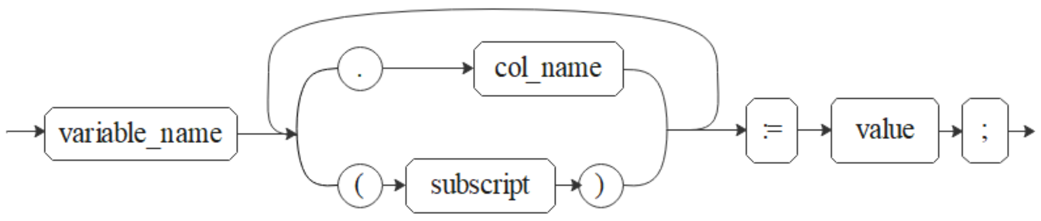
Example:

```
openGauss=# DECLARE
    emp_id  INTEGER := 7788; -- Assignment
BEGIN
    emp_id := 5; -- Assignment
    emp_id := 5*7784;
END;
/
```

Nested Value Assignment

Figure 12-5 shows the syntax diagram for assigning a nested value to a variable.

Figure 12-5 nested_assignment_value::=



The syntax in **Figure 12-5** is described as follows:

- **variable_name**: variable name
- **col_name**: column name
- **subscript**: subscript, which is used for an array variable. The value can be a value or an expression and must be of the int type.
- **value**: value or expression. The type of **value** must be compatible with the type of **variable_name**.

Example:

```
openGauss=#CREATE TYPE o1 as (a int, b int);
openGauss=# DECLARE
```

```
    TYPE r1 is VARRAY(10) of o1;
    emp_id  r1;
BEGIN
    emp_id(1).a := 5;-- Assign a value.
    emp_id(1).b := 5*7784;
END;
/
```

NOTICE

- In INTO mode, values can be assigned only to the columns at the first layer. Two-dimensional or above arrays are not supported.
- When a nested column value is referenced, if an array subscript exists, only one parenthesis can exist in the first three layers of columns. You are advised to use square brackets to reference the subscript.

INTO/BULK COLLECT INTO

INTO and **BULK COLLECT INTO** store values returned by statements in a stored procedure to variables. **BULK COLLECT INTO** allows some or all returned values to be temporarily stored in an array.

Example:

```
openGauss=# DECLARE
    my_id integer;
BEGIN
    select id into my_id from customers limit 1; -- Assign a value.
END;
/

openGauss=# DECLARE
    type id_list is varray(6) of customers.id%type;
    id_arr id_list;
BEGIN
    select id bulk collect into id_arr from customers order by id DESC limit 20; -- Assign values in batches.
END;
/
```

NOTICE

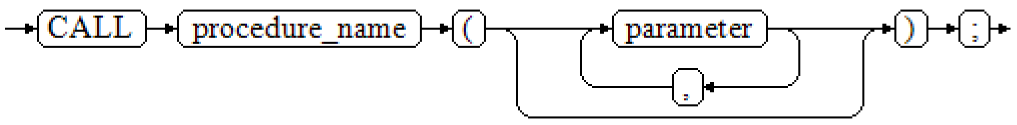
BULK COLLECT INTO can only assign values to arrays in batches. Use **LIMIT** properly to prevent performance deterioration caused by excessive operations on data.

12.6.3 Call Statements

Syntax

Figure 12-6 shows the syntax diagram for calling a clause.

Figure 12-6 call_clause::=



The above syntax diagram is explained as follows:

- **procedure_name** specifies the name of a stored procedure.
- **parameter** specifies the parameters for the stored procedure. You can set no parameter or multiple parameters.

Examples

```
-- Create the stored procedure proc_staffs.
openGauss=# CREATE OR REPLACE PROCEDURE proc_staffs
(
section     NUMBER(6),
salary_sum out NUMBER(8,2),
staffs_count out INTEGER
)
IS
BEGIN
SELECT sum(salary), count(*) INTO salary_sum, staffs_count FROM hr.staffs where section_id = section;
END;
/

-- Invoke the stored procedure proc_return.
openGauss=# CALL proc_staffs(2,8,6);

-- Delete a stored procedure.
openGauss=# DROP PROCEDURE proc_staffs;
```

12.7 Dynamic Statements

12.7.1 Executing Dynamic Query Statements

You can perform dynamic queries GaussDB provides two modes: EXECUTE IMMEDIATE and OPEN FOR. **EXECUTE IMMEDIATE** dynamically executes **SELECT** statements and **OPEN FOR** combines use of cursors. If you need to store query results in a data set, use **OPEN FOR**.

EXECUTE IMMEDIATE

Figure 12-7 shows the syntax diagram.

Figure 12-7 EXECUTE IMMEDIATE dynamic_select_clause::=

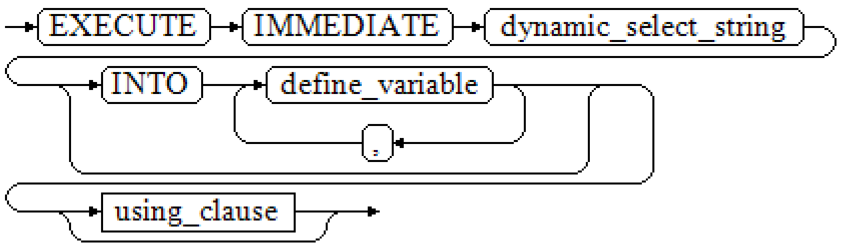
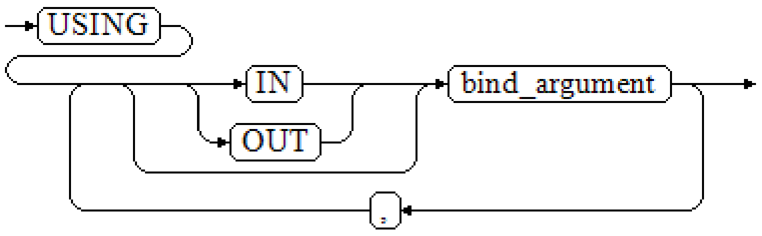


Figure 12-8 shows the syntax diagram for **using_clause**.

Figure 12-8 using_clause::=



The above syntax diagram is explained as follows:

- **define_variable**: specifies variables to store single-line query results.
- **USING IN bind_argument**: specifies where the variable passed to the dynamic SQL value is stored, that is, in the dynamic placeholder of **dynamic_select_string**.
- **USING OUT bind_argument**: specifies where the dynamic SQL returns the value of the variable.

NOTICE

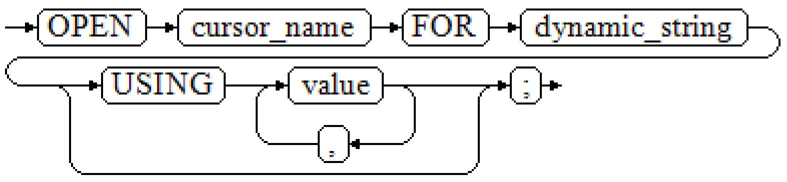
- In query statements, **INTO** and **OUT** cannot coexist.
- A placeholder name starts with a colon (:) followed by digits, characters, or strings, corresponding to *bind_argument* in the **USING** clause.
- *bind_argument* can only be a value, variable, or expression. It cannot be a database object such as a table name, column name, and data type. That is, *bind_argument* cannot be used to transfer schema objects for dynamic SQL statements. If a stored procedure needs to transfer database objects through *bind_argument* to construct dynamic SQL statements (generally, DDL statements), you are advised to use double vertical bars (||) to concatenate *dynamic_select_clause* with a database object.
- A dynamic PL/SQL block allows duplicate placeholders. That is, a placeholder can correspond to only one *bind_argument* in the **USING** clause.

OPEN FOR

Dynamic query statements can be executed by using **OPEN FOR** to open dynamic cursors.

Figure 12-9 shows the syntax diagram.

Figure 12-9 open_for::=



Parameter description:

- **cursor_name**: specifies the name of the cursor to be opened.
- **dynamic_string**: specifies the dynamic query statement.
- **USING** *value*: applies when a placeholder exists in dynamic_string.

For use of cursors, see **Cursors**.

12.7.2 Executing Dynamic Non-query Statements

Syntax

Figure 12-10 shows the syntax diagram.

Figure 12-10 noselect::=

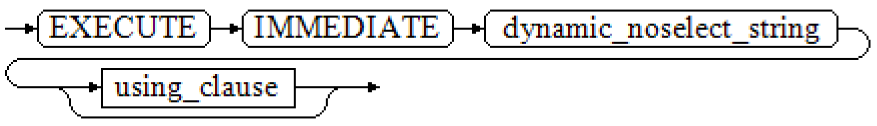
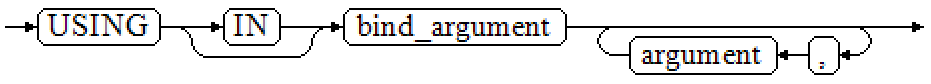


Figure 12-11 shows the syntax diagram for **using_clause**.

Figure 12-11 using_clause::=



The above syntax diagram is explained as follows:

USING IN *bind_argument* is used to specify the variable whose value is passed to the dynamic SQL statement. The variable is used when a placeholder exists in *dynamic_noselect_string*. That is, a placeholder is replaced by the corresponding *bind_argument* when a dynamic SQL statement is executed. Note that *bind_argument* can only be a value, variable, or expression, and cannot be a database object such as a table name, column name, and data type. If a stored procedure needs to transfer database objects through *bind_argument* to construct dynamic SQL statements (generally, DDL statements), you are advised to use double vertical bars (||) to concatenate *dynamic_select_clause* with a database object. In addition, a dynamic PL/SQL block allows duplicate placeholders. That is, a placeholder can correspond to only one *bind_argument*.

Examples

```
-- Create a table.
openGauss=# CREATE TABLE sections_t1
(
   section       NUMBER(4) ,
   section_name  VARCHAR2(30),
   manager_id    NUMBER(6),
   place_id      NUMBER(4)
);

-- Declare a variable.
openGauss=# DECLARE
   section       NUMBER(4) := 280;
   section_name  VARCHAR2(30) := 'Info support';
   manager_id    NUMBER(6) := 103;
   place_id      NUMBER(4) := 1400;
   new_colname   VARCHAR2(10) := 'sec_name';
BEGIN
-- Execute the query.
    EXECUTE IMMEDIATE 'insert into sections_t1 values(:1, :2, :3, :4)'
       USING section, section_name, manager_id,place_id;
-- Execute the query (duplicate placeholders).
    EXECUTE IMMEDIATE 'insert into sections_t1 values(:1, :2, :3, :1)'
       USING section, section_name, manager_id;
-- Run the ALTER statement. (You are advised to use double vertical bars (||) to concatenate the dynamic
DDL statement with a database object.)
    EXECUTE IMMEDIATE 'alter table sections_t1 rename section_name to ' || new_colname;
END;
/

-- Query data.
openGauss=# SELECT * FROM sections_t1;

-- Delete the table.
openGauss=# DROP TABLE sections_t1;
```

12.7.3 Dynamically Calling Stored Procedures

This section describes how to dynamically call store procedures. You must use anonymous statement blocks to package stored procedures or statement blocks and append **IN** and **OUT** behind the **EXECUTE IMMEDIATE...USING** statement to input and output parameters.

Syntax

Figure 12-12 shows the syntax diagram.

Figure 12-12 call_procedure::=

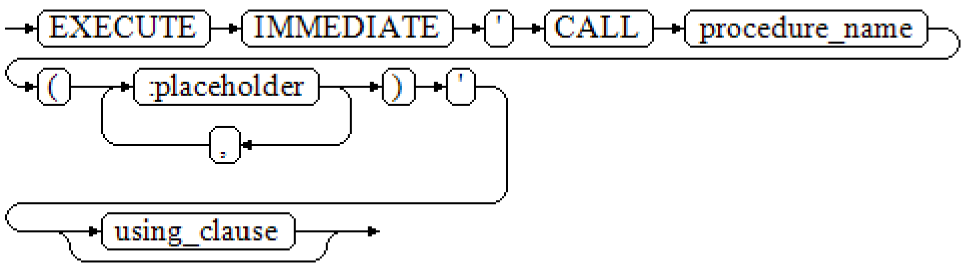
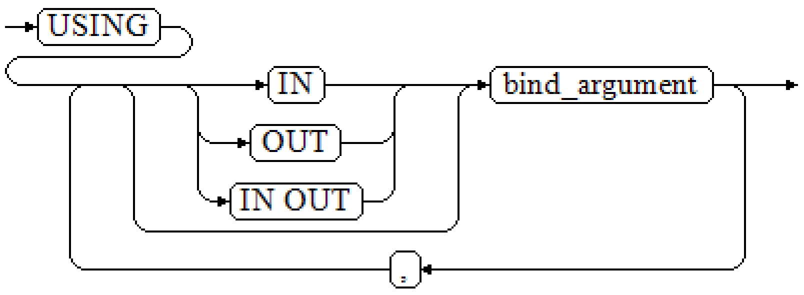


Figure 12-13 shows the syntax diagram for **using_clause**.

Figure 12-13 using_clause::=



The above syntax diagram is explained as follows:

- **CALL procedure_name**: calls the stored procedure.
- **[:placeholder1,:placeholder2,...]**: specifies the placeholder list of the stored procedure parameters. The numbers of the placeholders and parameters are the same.
- **USING [IN|OUT|IN OUT]bind_argument**: specifies where the variable passed to the stored procedure parameter value is stored. The modifiers in front of **bind_argument** and of the corresponding parameter are the same.

12.7.4 Dynamically Calling Anonymous Blocks

This section describes how to execute anonymous blocks in dynamic statements. Append **IN** and **OUT** behind the **EXECUTE IMMEDIATE...USING** statement to input and output parameters.

Syntax

Figure 12-14 shows the syntax diagram.

Figure 12-14 call_anonymous_block::=

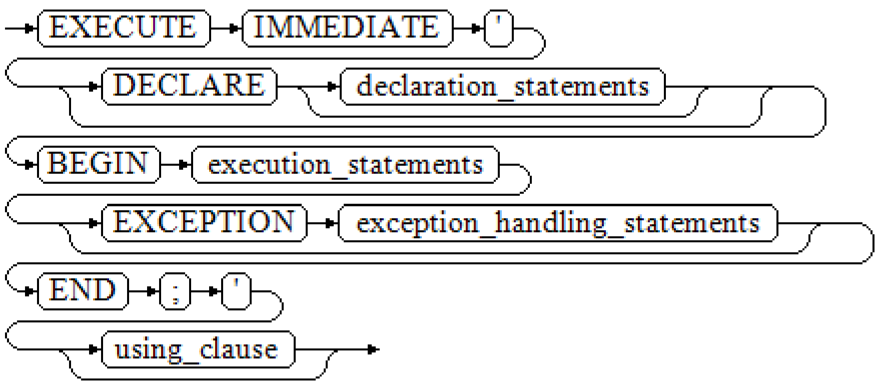
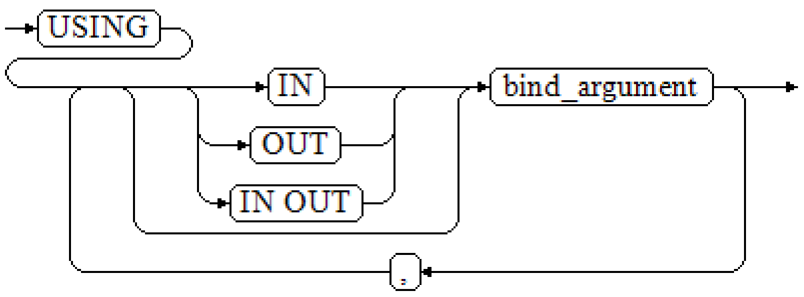


Figure 12-15 shows the syntax diagram for **using_clause**.

Figure 12-15 using_clause::=



The above syntax diagram is explained as follows:

- The execute part of an anonymous block starts with a **BEGIN** statement, has a break with an **END** statement, and ends with a semicolon (;).
- **USING [IN|OUT|IN OUT]bind_argument**: specifies where the variable passed to the stored procedure parameter value is stored. The modifiers in front of **bind_argument** and of the corresponding parameter are the same.
- The input and output parameters in the middle of an anonymous block are designated by placeholders. The numbers of the placeholders and parameters are the same. The sequences of the parameters corresponding to the placeholders and the USING parameters are the same.
- Currently in GaussDB, when dynamic statements call anonymous blocks, placeholders cannot be used to pass input and output parameters in an **EXCEPTION** statement.

12.8 Control Statements

12.8.1 RETURN Statements

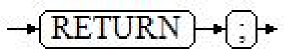
In GaussDB, data can be returned in either of the following ways: **RETURN**, **RETURN NEXT**, or **RETURN QUERY**. **RETURN NEXT** and **RETURN QUERY** are used only for functions and cannot be used for stored procedures.

12.8.1.1 RETURN

Syntax

Figure 12-16 shows the syntax diagram for a return statement.

Figure 12-16 return_clause::=



The above syntax diagram is explained as follows:

This statement returns control from a stored procedure or function to a caller.

Examples

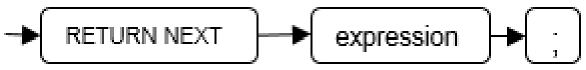
See **Examples** for call statement examples.

12.8.1.2 RETURN NEXT and RETURN QUERY

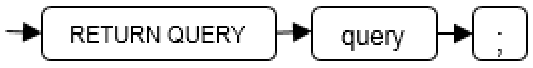
Syntax

When creating a function, specify **SETOF** *datatype* for the return values.

return_next_clause::=



return_query_clause::=



The above syntax diagram is explained as follows:

If a function needs to return a result set, use **RETURN NEXT** or **RETURN QUERY** to add results to the result set, and then continue to execute the next statement of the function. As the **RETURN NEXT** or **RETURN QUERY** statement is executed repeatedly, more and more results will be added to the result set. After the function is executed, all results are returned.

RETURN NEXT can be used for scalar and compound data types.

RETURN QUERY has a variant **RETURN QUERY EXECUTE**. You can add dynamic queries and add parameters to the queries by **USING**.

Examples

```
openGauss=# CREATE TABLE t1(a int);
openGauss=# INSERT INTO t1 VALUES(1),(10);

--RETURN NEXT
openGauss=# CREATE OR REPLACE FUNCTION fun_for_return_next() RETURNS SETOF t1 AS $$
DECLARE
   r t1%ROWTYPE;
BEGIN
   FOR r IN select * from t1
   LOOP
      RETURN NEXT r;
   END LOOP;
   RETURN;
END;
$$ LANGUAGE PLPGSQL;
openGauss=# call fun_for_return_next();
 a
---
 1
 10
(2 rows)

-- RETURN QUERY
openGauss=# CREATE OR REPLACE FUNCTION fun_for_return_query() RETURNS SETOF t1 AS $$
DECLARE
   r t1%ROWTYPE;
BEGIN
   RETURN QUERY select * from t1;
END;
$$
language plpgsql;
openGauss=# call fun_for_return_query();
 a
---
 1
 10
(2 rows)
```

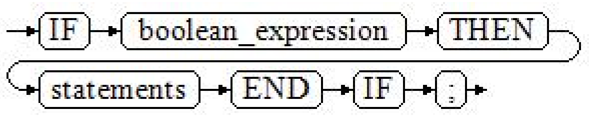
12.8.2 Conditional Statements

Conditional statements are used to decide whether given conditions are met. Operations are executed based on the decisions made.

GaussDB supports five usages of **IF**:

- IF_THEN

Figure 12-17 IF_THEN::=



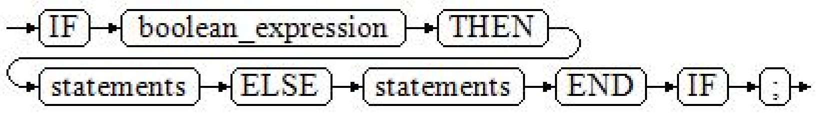
IF_THEN is the simplest form of **IF**. If the condition is true, statements are executed. If it is false, they are skipped.

Example:

```
openGauss=# IF v_user_id <> 0 THEN
    UPDATE users SET email = v_email WHERE user_id = v_user_id;
END IF;
```

- IF_THEN_ELSE

  Figure 12-18 IF_THEN_ELSE::=

  IF_THEN_ELSE statements add **ELSE** branches and can be executed if the condition is false.

  Example:

```
openGauss=# IF parentid IS NULL OR parentid = ''
THEN
    RETURN;
ELSE
    hp_true_filename(parentid); -- Call the stored procedure.
END IF;
```

- IF_THEN_ELSE IF

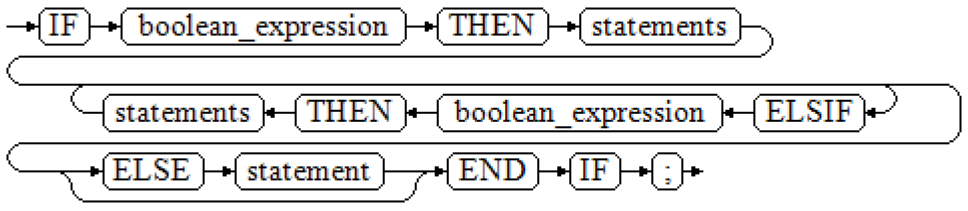
  IF statements can be nested in the following way:

```
openGauss=# IF sex = 'm' THEN
    pretty_sex := 'man';
ELSE
    IF sex = 'f' THEN
        pretty_sex := 'woman';
    END IF;
END IF;
```

  Actually, this is a way of an **IF** statement nesting in the **ELSE** part of another **IF** statement. Therefore, an **END IF** statement is required for each nesting **IF** statement and another **END IF** statement is required to end the parent **IF-ELSE** statement. To set multiple options, use the following form:

- IF_THEN_ELSIF_ELSE

  Figure 12-19 IF_THEN_ELSIF_ELSE::=

  Example:

```
IF number_tmp = 0 THEN
    result := 'zero';
ELSIF number_tmp > 0 THEN
    result := 'positive';
ELSIF number_tmp < 0 THEN
    result := 'negative';
ELSE
    result := 'NULL';
END IF;
```

- IF_THEN_ELSEIF_ELSE

  ELSEIF is an alias of **ELSIF**.

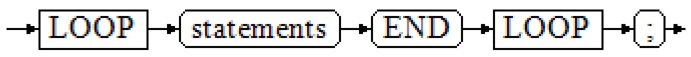
Example:
```
CREATE OR REPLACE PROCEDURE proc_control_structure(i in integer)
AS
    BEGIN
        IF i > 0 THEN
            raise info 'i:% is greater than 0. ',i;
        ELSIF i < 0 THEN
            raise info 'i:% is smaller than 0. ',i;
        ELSE
            raise info 'i:% is equal to 0. ',i;
        END IF;
        RETURN;
    END;
/

CALL proc_control_structure(3);

-- Delete the stored procedure.
DROP PROCEDURE proc_control_structure;
```

12.8.3 Loop Statements

Simple LOOP Statements

Syntax diagram

Figure 12-20 loop::=



Example

```
CREATE OR REPLACE PROCEDURE proc_loop(i in integer, count out integer)
AS
    BEGIN
        count:=0;
        LOOP
        IF count > i THEN
            raise info 'count is %. ', count;
            EXIT;
        ELSE
            count:=count+1;
        END IF;
        END LOOP;
    END;
/

CALL proc_loop(10,5);
```

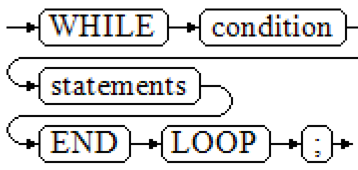
NOTICE

The loop must be exploited together with **EXIT**; otherwise, a dead loop occurs.

WHILE_LOOP Statements

Syntax diagram

Figure 12-21 while_loop::=



If the conditional expression is true, a series of statements in the WHILE statement are repeatedly executed and the condition is decided each time the loop body is executed.

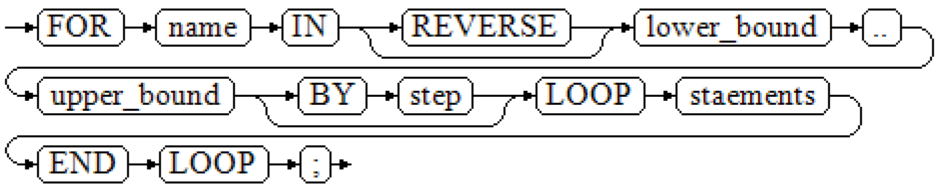
Example

```
CREATE TABLE integertable(c1 integer) ;
CREATE OR REPLACE PROCEDURE proc_while_loop(maxval in integer)
AS
    DECLARE
    i int :=1;
    BEGIN
        WHILE i < maxval LOOP
            INSERT INTO integertable VALUES(i);
            i:=i+1;
        END LOOP;
    END;
/

-- Invoke a function.
CALL proc_while_loop(10);

-- Delete the stored procedure and table.
DROP PROCEDURE proc_while_loop;
DROP TABLE integertable;
```

FOR_LOOP (*Integer variable*) *Statement*

Syntax diagram

Figure 12-22 for_loop::=



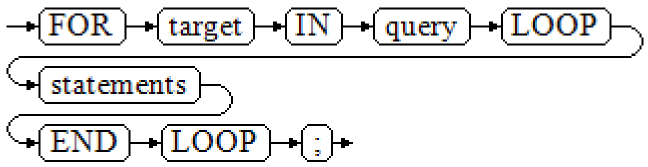
NOTE

- The variable *name* is automatically defined as the integer type and exists only in this loop. The value of *name* ranges from **lower_bound** to **upper_bound**.
- When the keyword **REVERSE** is used, the value of **lower_bound** must be greater than or equal to that of **upper_bound**; otherwise, the loop body is not executed.

FOR_LOOP Query Statements

Syntax diagram

Figure 12-23 for_loop_query::=



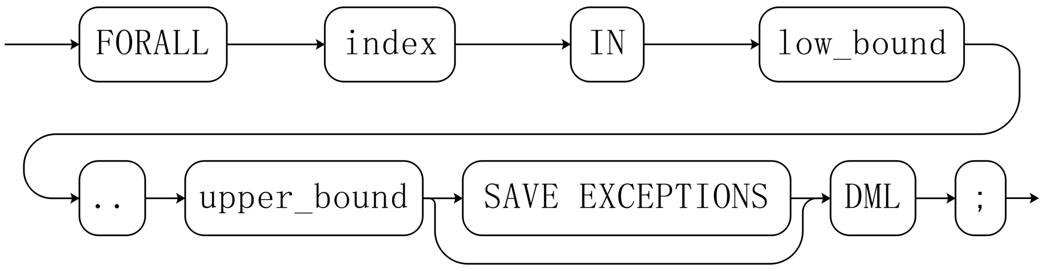
NOTE

The variable *target* is automatically defined, its type is the same as that in the query result, and it is valid only in this loop. The value of *target* is the query result.

FORALL Batch Query Statements

Syntax diagram

Figure 12-24 forall::=



NOTE

- The variable *index* is automatically defined as the integer type and exists only in this loop. The value of *index* falls between the value of **low_bound** and the value of **upper_bound**.
- If **SAVE EXCEPTIONS** is specified, exceptions occurred during DML execution in the loop body are saved in **SQL&BULK_EXCEPTIONS** and an exception is thrown after the execution is complete. If there is no abnormal execution result in the loop, the loop will not be rolled back in the current subtransaction.

Example

```
CREATE TABLE hdfs_t1 (
  title NUMBER(6),
  did VARCHAR2(20),
  data_period VARCHAR2(25),
  kind VARCHAR2(25),
  interval VARCHAR2(20),
  time DATE,
  isModified VARCHAR2(10)
);

INSERT INTO hdfs_t1 VALUES( 8, 'Donald', 'OConnell', 'DOCONNEL', '650.507.9833', to_date('21-06-1999',
'dd-mm-yyyy'), 'SH_CLERK' );

CREATE OR REPLACE PROCEDURE proc_forall()
AS
BEGIN
    FORALL i IN 100..120
        update hdfs_t1 set title = title + 100*i;
END;
/

-- Invoke a function.
CALL proc_forall();

-- Query the invocation result of the stored procedure.
SELECT * FROM hdfs_t1 WHERE title BETWEEN 100 AND 120;

-- Delete the stored procedure and table.
DROP PROCEDURE proc_forall;
DROP TABLE hdfs_t1;
```

12.8.4 Branch Statements

Syntax

Figure 12-25 shows the syntax diagram for a branch statement.

Figure 12-25 case_when::=

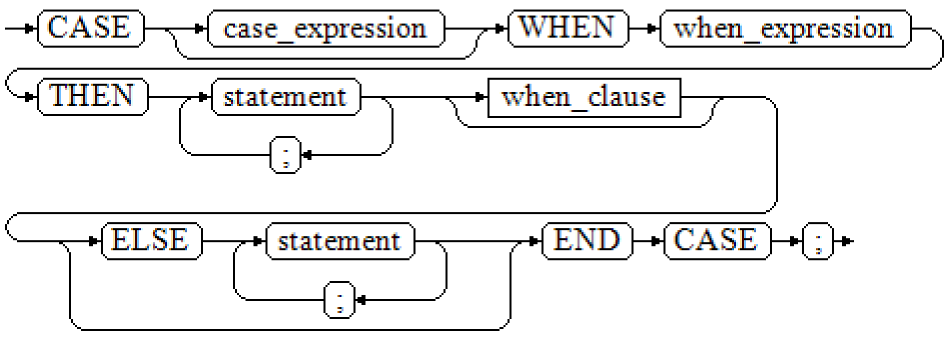
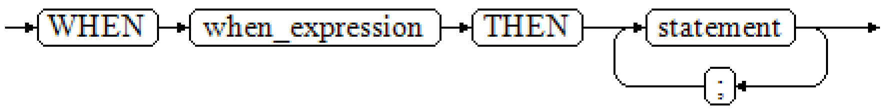


Figure 12-26 shows the syntax diagram for **when_clause**.

Figure 12-26 when_clause::=



Parameter description:

- *case_expression*: specifies the variable or expression.
- *when_expression*: specifies the constant or conditional expression.
- *statement*: specifies the statement to be executed.

Examples

```
CREATE OR REPLACE PROCEDURE proc_case_branch(pi_result in integer, pi_return out integer)
AS
    BEGIN
        CASE pi_result
            WHEN 1 THEN
                pi_return := 111;
            WHEN 2 THEN
                pi_return := 222;
            WHEN 3 THEN
                pi_return := 333;
            WHEN 6 THEN
                pi_return := 444;
            WHEN 7 THEN
                pi_return := 555;
            WHEN 8 THEN
                pi_return := 666;
            WHEN 9 THEN
                pi_return := 777;
            WHEN 10 THEN
                pi_return := 888;
            ELSE
                pi_return := 999;
        END CASE;
        raise info 'pi_return : %',pi_return ;
END;
/

CALL proc_case_branch(3,0);

-- Delete the stored procedure.
DROP PROCEDURE proc_case_branch;
```

12.8.5 NULL Statements

In PL/SQL programs, **NULL** statements are used to indicate "nothing should be done", equal to placeholders. They grant meanings to some statements and improve program readability.

Syntax

The following shows example use of **NULL** statements.

```
DECLARE
    ...
BEGIN
    ...
    IF v_num IS NULL THEN
        NULL; --No data needs to be processed.
    END IF;
END;
/
```

12.8.6 Error Trapping Statements

By default, any error occurring in a PL/SQL function aborts execution of the function, and indeed of the surrounding transaction as well. You can trap errors and restore from them by using a **BEGIN** block with an **EXCEPTION** clause. The syntax is an extension of the normal syntax for a **BEGIN** block:

```
[<<label>>]
[DECLARE
    declarations]
BEGIN
    statements
EXCEPTION
    WHEN condition [OR condition ...] THEN
        handler_statements
    [WHEN condition [OR condition ...] THEN
        handler_statements
    ...]
END;
```

If no error occurs, this form of block simply executes all the statements, and then control passes to the next statement after **END**. But if an error occurs within the statements, further processing of the statements is abandoned, and control passes to the **EXCEPTION** list. The list is searched for the first condition matching the error that occurred. If a match is found, the corresponding **handler_statements** are executed, and then control passes to the next statement after **END**. If no match is found, the error propagates out as though the **EXCEPTION** clause were not there at all: Error codes can be used to catch other error codes of the same type.

The error can be caught by an enclosing block with **EXCEPTION**, or if there is none it aborts processing of the function.

The condition names can be any of those shown in SQL standard error codes. The special condition name **OTHERS** matches every error type except **QUERY_CANCELED**.

If a new error occurs within the selected **handler_statements**, it cannot be caught by this **EXCEPTION** clause, but is propagated out. A surrounding **EXCEPTION** clause could catch it.

When an error is caught by an **EXCEPTION** clause, the local variables of the PL/SQL function remain as they were when the error occurred, but all changes to persistent database state within the block are rolled back.

Example:

```
CREATE TABLE mytab(id INT,firstname VARCHAR(20),lastname VARCHAR(20)) ;

INSERT INTO mytab(firstname, lastname) VALUES('Tom', 'Jones');

CREATE FUNCTION fun_exp() RETURNS INT
AS $$
DECLARE
    x INT :=0;
    y INT;
BEGIN
    UPDATE mytab SET firstname = 'Joe' WHERE lastname = 'Jones';
    x := x + 1;
    y := x / 0;
EXCEPTION
    WHEN division_by_zero THEN
        RAISE NOTICE 'caught division_by_zero';
        RETURN x;
END;$$
LANGUAGE plpgsql;

call fun_exp();
NOTICE:  caught division_by_zero
```

```
 fun_exp
---------
       1
(1 row)

select * from mytab;
 id | firstname | lastname
----+-----------+----------
    | Tom       | Jones
(1 row)

DROP FUNCTION fun_exp();
DROP TABLE mytab;
```

When control reaches the assignment to **y**, it will fail with a **division_by_zero** error. This will be caught by the **EXCEPTION** clause. The value returned in the **RETURN** statement will be the incremented value of **x**.

NOTE

A block containing an **EXCEPTION** clause is more expensive to enter and exit than a block without one. Therefore, do not use **EXCEPTION** without need.

In the following scenario, an exception cannot be caught, and the entire transaction rolls back. The threads of the nodes participating the stored procedure exit abnormally due to node failure and network fault, or the source data is inconsistent with that of the table structure of the target table during the COPY FROM operation.

Example: Exceptions with **UPDATE**/**INSERT**

This example uses exception handling to perform either **UPDATE** or **INSERT**, as appropriate:

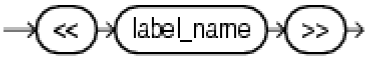
```
CREATE TABLE db (a INT, b TEXT);

CREATE FUNCTION merge_db(key INT, data TEXT) RETURNS VOID AS
$$
BEGIN
    LOOP

-- First try to update the key
        UPDATE db SET b = data WHERE a = key;
        IF found THEN
            RETURN;
        END IF;
-- Not there, so try to insert the key. If someone else inserts the same key concurrently, we could get a
unique-key failure.
        BEGIN
            INSERT INTO db(a,b) VALUES (key, data);
            RETURN;
        EXCEPTION WHEN unique_violation THEN
        -- Do nothing, and loop to try the UPDATE again.
        END;
     END LOOP;
END;
$$
LANGUAGE plpgsql;

SELECT merge_db(1, 'david');
SELECT merge_db(1, 'dennis');

--Delete FUNCTION and TABLE:
DROP FUNCTION merge_db;
DROP TABLE db ;
```

12.8.7 GOTO Statements

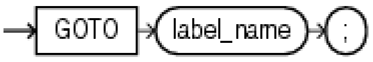
A **GOTO** statement unconditionally transfers the control from the current statement to a labeled statement. The **GOTO** statement changes the execution logic. Therefore, use this statement only when necessary. Alternatively, you can use the **EXCEPTION** statement to handle issues in special scenarios. To run a **GOTO** statement, the labeled statement must be unique.

Syntax

label declaration ::=



goto statement ::=



Examples

```
openGauss=# CREATE OR REPLACE PROCEDURE GOTO_test()
AS
DECLARE
    v1  int;
BEGIN
    v1  := 0;
        LOOP
        EXIT WHEN v1 > 100;
                v1 := v1 + 2;
                if v1 > 25 THEN
                        GOTO pos1;
                END IF;
        END LOOP;
<<pos1>>
v1 := v1 + 10;
raise info 'v1 is %. ', v1;
END;
/

call GOTO_test();
```

Constraints

Using **GOTO** statements has the following constraints:

- A **GOTO** statement does not allow multiple labeled statements even if the statements are in different blocks.

```
BEGIN
  GOTO pos1;
  <<pos1>>
  SELECT * FROM ...
  <<pos1>>
  UPDATE t1 SET ...
END;
```

- A **GOTO** statement cannot transfer control to the **IF**, **CASE**, or **LOOP** statement.

```
BEGIN
   GOTO pos1;
   IF valid THEN
     <<pos1>>
```

```
    SELECT * FROM ...
  END IF;
 END;
```

- A **GOTO** statement cannot transfer control from one **IF** clause to another, or from one **WHEN** clause in the **CASE** statement to another.

```
BEGIN
  IF valid THEN
    GOTO pos1;
    SELECT * FROM ...
  ELSE
    <<pos1>>
    UPDATE t1 SET ...
  END IF;
 END;
```

- A **GOTO** statement cannot transfer control from an outer block to an inner **BEGIN-END** block.

```
BEGIN
  GOTO pos1;
  BEGIN
    <<pos1>>
    UPDATE t1 SET ...
  END;
 END;
```

- A **GOTO** statement cannot transfer control from an exception handler to the current **BEGIN-END** block. However, a **GOTO** statement can transfer control to the upper-layer **BEGIN-END** block.

```
BEGIN
  <<pos1>>
  UPDATE t1 SET ...
  EXCEPTION
    WHEN condition THEN
       GOTO pos1;
 END;
```

- To branch to a position that does not have an executable statement, add the **NULL** statement.

```
DECLARE
  done  BOOLEAN;
BEGIN
  FOR i IN 1..50 LOOP
     IF done THEN
        GOTO end_loop;
     END IF;
     <<end_loop>>  -- not allowed unless an executable statement follows
     NULL; -- add NULL statement to avoid error
  END LOOP;  -- raises an error without the previous NULL
END;
/
```

12.9 Transaction Management

Calling a stored procedure automatically starts a transaction. When the calling is complete, the transaction is automatically committed, or rolled back upon an exception. In addition to automatic transaction control, you can also use COMMIT/ ROLLBACK to control transactions in stored procedures. Running the COMMIT/ ROLLBACK commands in a stored procedure will commit or roll back the current transaction and automatically starts a new transaction. All subsequent operations will be performed in the new transaction.

A savepoint is a special mark inside a transaction. It allows all commands that are executed after it was established to be rolled back, restoring the transaction state to what it was at the time of the savepoint. In a stored procedure, you can use savepoints to manage transactions. Currently, you can create, roll back, and release savepoints. If a savepoint for rollback is used in a stored procedure, only the modification of the current transaction is rolled back. The execution process of the stored procedure is not changed, and the values of local variables in the stored procedure are not rolled back.

Syntax

```
Define a savepoint.
   SAVEPOINT savepoint_name;
Roll back a savepoint.
   ROLLBACK TO [SAVEPOINT] savepoint_name;
Release a savepoint.
   RELEASE [SAVEPOINT] savepoint_name;
```

Usage Scenarios

The applicable contexts are as follows:

- COMMIT/ROLLBACK/SAVEPOINT can be used in stored procedures in PL/SQL.
- COMMIT, ROLLBACK, and SAVEPOINT can be used in stored procedures that contain EXCEPTION.
- COMMIT, ROLLBACK, and SAVEPOINT can be used in EXCEPTION statements of stored procedures.
- A stored procedure that contains COMMIT, ROLLBACK, or SAVEPOINT (which means the stored procedure is controlled by BEGIN, START, or END) can be called in a transaction block.
- A stored procedure that contains savepoints can be invoked in a subtransaction. That is, an externally defined savepoint is used in the stored procedure to roll back the transaction to the savepoint defined outside the stored procedure.
- A stored procedure is visible to a savepoint defined in the stored procedure. That is, the modification of the transaction can be rolled back to the savepoint defined in the stored procedure.
- COMMIT, ROLLBACK, and SAVEPOINT, as well as IF, FOR, CURSOR LOOP, and WHILE, can be called in most contexts and statements in PLSQL.
- A stored procedure or function that contains COMMIT, ROLLBACK, or SAVEPOINT can be called in the return values and simple expression calculation of stored procedures.

The following content can be committed or rolled back:

- DDL statements after COMMIT/ROLLBACK can be committed or rolled back.
- DML statements after COMMIT/ROLLBACK can be committed.
- GUC parameters in stored procedures can be committed or rolled back.

Usage Restrictions

COMMIT and ROLLBACK cannot be used in the following contexts:

- COMMIT, ROLLBACK, and SAVEPOINT cannot be called in stored procedures other than PLSQL, such as PLJava and PLPython.
- COMMIT, ROLLBACK, SAVEPOINT and stored procedures that contain COMMIT, ROLLBACK, or SAVEPOINT cannot be called in functions.

- After SAVEPOINT is called in a transaction block, stored procedures that contain COMMIT/ROLLBACK cannot be called.
- Stored procedures that contain COMMIT, ROLLBACK, or SAVEPOINT cannot be called in TRIGGER.
- COMMIT, ROLLBACK, and SAVEPOINT cannot be invoked in EXECUTE statements.
- Stored procedures that contain COMMIT, ROLLBACK, or SAVEPOINT cannot be called in CURSOR statements.
- Stored procedures that contain IMMUTABLE or SHIPPABLE cannot call COMMIT, ROLLBACK, SAVEPOINT or another stored procedure that contain COMMIT, ROLLBACK, or SAVEPOINT.
- Stored procedures that contain COMMIT, ROLLBACK, or SAVEPOINT cannot be called in SQL statements other than SELECT PROC and CALL PROC.
- COMMIT, ROLLBACK, or SAVEPOINT cannot be called in a stored procedure whose header contains GUC parameters.
- COMMIT, ROLLBACK, or SAVEPOINT cannot be called in expressions or CURSOR and EXECUTE statements.
- An autonomous transaction and a stored procedure transaction are two independent transactions that cannot use the savepoints defined in each other.

The following content cannot be committed or rolled back:

- Variables declared or imported in stored procedures cannot be committed or rolled back.
- In stored procedures, GUC parameters that take effect only after a restart cannot be committed or rolled back.

Examples

- Example 1: COMMIT/ROLLBACK can be used in stored procedures in PLSQL.

```
CREATE TABLE EXAMPLE1(COL1 INT);

CREATE OR REPLACE PROCEDURE TRANSACTION_EXAMPLE()
AS
BEGIN
    FOR i IN 0..20 LOOP
        INSERT INTO EXAMPLE1(COL1) VALUES (i);
        IF i % 2 = 0 THEN
            COMMIT;
        ELSE
            ROLLBACK;
        END IF;
    END LOOP;
END;
/
```

- Example 2:

  COMMIT and ROLLBACK can be used in stored procedures that contain EXCEPTION.

  COMMIT and ROLLBACK can be used in EXCEPTION statements of stored procedures.

  DDL statements after COMMIT or ROLLBACK can be committed or rolled back.

```
CREATE OR REPLACE PROCEDURE TEST_COMMIT_INSERT_EXCEPTION_ROLLBACK()
AS
BEGIN
 DROP TABLE IF EXISTS TEST_COMMIT;
 CREATE TABLE TEST_COMMIT(A INT, B INT);
 INSERT INTO TEST_COMMIT SELECT 1, 1;
 COMMIT;
        CREATE TABLE TEST_ROLLBACK(A INT, B INT);
 RAISE EXCEPTION 'RAISE EXCEPTION AFTER COMMIT';
EXCEPTION
    WHEN OTHERS THEN
 INSERT INTO TEST_COMMIT SELECT 2, 2;
 ROLLBACK;
END;
/
```

- Example 3: A stored procedure that contains COMMIT or ROLLBACK (which means the stored procedure is controlled by BEGIN, START, or END) can be called in a transaction block.

```
BEGIN;
    CALL TEST_COMMIT_INSERT_EXCEPTION_ROLLBACK();
END;
```

- Example 4: COMMIT/ROLLBACK, including IF, FOR, CURSOR LOOP, and WHILE, can be called in most PLSQL contexts and statements.

```
CREATE OR REPLACE PROCEDURE TEST_COMMIT2()
IS
BEGIN
    DROP TABLE IF EXISTS TEST_COMMIT;
    CREATE TABLE TEST_COMMIT(A INT);
    FOR I IN REVERSE 3..0 LOOP
 INSERT INTO TEST_COMMIT SELECT I;
 COMMIT;
    END LOOP;
    FOR I IN REVERSE 2..4 LOOP
 UPDATE TEST_COMMIT SET A=I;
 COMMIT;
    END LOOP;
EXCEPTION
WHEN OTHERS THEN
 INSERT INTO TEST_COMMIT SELECT 4;
    COMMIT;
END;
/
```

- Example 5: Return values and simple expression calculation of stored procedures are supported.

```
CREATE OR REPLACE PROCEDURE exec_func3(RET_NUM OUT INT)
AS
BEGIN
    RET_NUM := 1+1;
COMMIT;
END;
/
CREATE OR REPLACE PROCEDURE exec_func4(ADD_NUM IN INT)
AS
SUM_NUM INT;
BEGIN
SUM_NUM := ADD_NUM + exec_func3();
COMMIT;
END;
/
```

- Example 6: GUC parameters in stored procedures can be rolled back to a commit.

```
SHOW explain_perf_mode;
SHOW enable_force_vector_engine;

CREATE OR REPLACE PROCEDURE GUC_ROLLBACK()
```

```
AS
BEGIN
    SET enable_force_vector_engine = on;
    COMMIT;
    SET explain_perf_mode TO pretty;
    ROLLBACK;
END;
/

call GUC_ROLLBACK();
SHOW explain_perf_mode;
SHOW enable_force_vector_engine;
SET enable_force_vector_engine = off;
```

- Example 7: COMMIT, ROLLBACK, and stored procedures that contain COMMIT or ROLLBACK cannot be called in functions.

```
CREATE OR REPLACE FUNCTION FUNCTION_EXAMPLE1() RETURN INT
AS
EXP INT;
BEGIN
    FOR i IN 0..20 LOOP
        INSERT INTO EXAMPLE1(col1) VALUES (i);
        IF i % 2 = 0 THEN
            COMMIT;
        ELSE
            ROLLBACK;
        END IF;
    END LOOP;
    SELECT COUNT(*) FROM EXAMPLE1 INTO EXP;
    RETURN EXP;
END;
/
```

- Example 8: Stored procedures that contain COMMIT or ROLLBACK cannot be called in functions.

```
CREATE OR REPLACE FUNCTION FUNCTION_EXAMPLE2() RETURN INT
AS
EXP INT;
BEGIN
    -- transaction_example is a stored procedure and contains the COMMIT/ROLLBACK statement.
    CALL transaction_example();
    SELECT COUNT(*) FROM EXAMPLE1 INTO EXP;
    RETURN EXP;
END;
/
```

- Example 9: A TRIGGER stored procedure cannot contain COMMIT or ROLLBACK or call another stored procedure that contains COMMIT or ROLLBACK.

```
CREATE OR REPLACE FUNCTION FUNCTION_TRI_EXAMPLE2() RETURN TRIGGER
AS
EXP INT;
BEGIN
    FOR i IN 0..20 LOOP
        INSERT INTO EXAMPLE1(col1) VALUES (i);
        IF i % 2 = 0 THEN
            COMMIT;
        ELSE
            ROLLBACK;
        END IF;
    END LOOP;
    SELECT COUNT(*) FROM EXAMPLE1 INTO EXP;
END;
/

CREATE TRIGGER TRIGGER_EXAMPLE AFTER DELETE ON EXAMPLE1
FOR EACH ROW EXECUTE PROCEDURE FUNCTION_TRI_EXAMPLE2();

DELETE FROM EXAMPLE1;
```

- Example 10: Stored procedures that contain IMMUTABLE or SHIPPABLE cannot call COMMIT, ROLLBACK, or another stored procedure that contains COMMIT or ROLLBACK.

```
CREATE OR REPLACE PROCEDURE TRANSACTION_EXAMPLE1()
IMMUTABLE
AS
BEGIN
    FOR i IN 0..20 LOOP
        INSERT INTO EXAMPLE1 (col1) VALUES (i);
        IF i % 2 = 0 THEN
            COMMIT;
        ELSE
            ROLLBACK;
        END IF;
    END LOOP;
END;
/
```

- Example 11: Variables declared or passed in stored procedures cannot be committed.

```
CREATE OR REPLACE PROCEDURE TRANSACTION_EXAMPLE2(EXP_OUT OUT INT)
AS
EXP INT;
BEGIN
    EXP_OUT := 0;
    COMMIT;
    DBE_OUTPUT.PRINT_LINE('EXP IS:'||EXP);
    EXP_OUT := 1;
    ROLLBACK;
    DBE_OUTPUT.PRINT_LINE('EXP IS:'||EXP);
END;
/
```

- Example 12: Calling in SQL statements (other than Select Procedure) is not supported.

```
CREATE OR REPLACE PROCEDURE TRANSACTION_EXAMPLE3()
AS
BEGIN
    FOR i IN 0..20 LOOP
        INSERT INTO EXAMPLE1 (col1) VALUES (i);
        IF i % 2 = 0 THEN
            EXECUTE IMMEDIATE 'COMMIT';
        ELSE
            EXECUTE IMMEDIATE 'ROLLBACK';
        END IF;
    END LOOP;
END;
/
```

- Example 13: COMMIT or ROLLBACK cannot be called in a stored procedure whose header contains GUC parameters.

```
CREATE OR REPLACE PROCEDURE TRANSACTION_EXAMPLE4()
SET ARRAY_NULLS TO "ON"
AS
BEGIN
    FOR i IN 0..20 LOOP
        INSERT INTO EXAMPLE1 (col1) VALUES (i);
        IF i % 2 = 0 THEN
            COMMIT;
        ELSE
            ROLLBACK;
        END IF;
    END LOOP;
END;
/
```

- Example 14: A stored procedure whose cursor is open cannot contain COMMIT or ROLLBACK.

```
CREATE OR REPLACE PROCEDURE TRANSACTION_EXAMPLE5(INTIN IN INT, INTOUT OUT INT)
AS
BEGIN
INTOUT := INTIN + 1;
COMMIT;
END;
/

CREATE OR REPLACE PROCEDURE TRANSACTION_EXAMPLE6()
AS
CURSOR CURSOR1(EXPIN INT)
IS SELECT TRANSACTION_EXAMPLE5(EXPIN);
INTEXP INT;
BEGIN
    FOR i IN 0..20 LOOP
        OPEN CURSOR1(i);
        FETCH CURSOR1 INTO INTEXP;
        INSERT INTO EXAMPLE1(COL1) VALUES (INTEXP);
        IF i % 2 = 0 THEN
            COMMIT;
        ELSE
            ROLLBACK;
        END IF;
        CLOSE CURSOR1;
    END LOOP;
END;
/
```

- Example 15: COMMIT or ROLLBACK cannot be called in expressions or CURSOR and EXECUTE statements.

```
CREATE OR REPLACE PROCEDURE exec_func1()
AS
BEGIN
    CREATE TABLE TEST_exec(A INT);
COMMIT;
END;
/
CREATE OR REPLACE PROCEDURE exec_func2()
AS
BEGIN
EXECUTE exec_func1();
COMMIT;
END;
/
```

- Example 16: Roll back some modifications of stored procedure on a transaction to a savepoint.

```
CREATE OR REPLACE PROCEDURE STP_SAVEPOINT_EXAMPLE1()
AS
BEGIN
    INSERT INTO EXAMPLE1 VALUES(1);
    SAVEPOINT s1;
    INSERT INTO EXAMPLE1 VALUES(2);
    ROLLBACK TO s1;  -- Roll back the insertion of record 2.
    INSERT INTO EXAMPLE1 VALUES(3);
END;
/
```

- Example 17: Roll back a stored procedure to a savepoint defined outside the stored procedure.

```
CREATE OR REPLACE PROCEDURE STP_SAVEPOINT_EXAMPLE2()
AS
BEGIN
    INSERT INTO EXAMPLE1 VALUES(2);
    ROLLBACK TO s1;  -- Roll back the insertion of record 2.
    INSERT INTO EXAMPLE1 VALUES(3);
END;
/
```

```
BEGIN;
INSERT INTO EXAMPLE1 VALUES(1);
SAVEPOINT s1;
CALL STP_SAVEPOINT_EXAMPLE2();
SELECT * FROM EXAMPLE1;
COMMIT;
```

- Example 18: Roll back an external SQL or other stored procedure to a savepoint defined in the stored procedure.

```
CREATE OR REPLACE PROCEDURE STP_SAVEPOINT_EXAMPLE3()
AS
BEGIN
    INSERT INTO EXAMPLE1 VALUES(1);
    SAVEPOINT s1;
    INSERT INTO EXAMPLE1 VALUES(2);
END;
/

BEGIN;
INSERT INTO EXAMPLE1 VALUES(3);
CALL STP_SAVEPOINT_EXAMPLE3();
ROLLBACK TO SAVEPOINT s1; -- Roll back the insertion of record 2 to the stored procedure.
SELECT * FROM EXAMPLE1;
COMMIT;
```

12.10 Other Statements

12.10.1 Lock Operations

GaussDB provides multiple lock modes to control concurrent accesses to table data. These modes are used when MVCC cannot give expected behaviors. Alike, most GaussDB commands automatically apply appropriate locks to ensure that called tables are not deleted or modified in an incompatible manner during command execution. For example, when concurrent operations exist, **ALTER TABLE** cannot be executed on the same table.

12.10.2 Cursor Operations

GaussDB provides cursors as a data buffer for users to store execution results of SQL statements. Each cursor region has a name. Users can use SQL statements to obtain records one by one from cursors and grant the records to master variables, then being processed further by host languages.

Cursor operations include cursor definition, open, fetch, and close operations.

For the complete example of cursor operations, see **Explicit Cursor**.

12.11 Cursors

12.11.1 Overview

To process SQL statements, the stored procedure process assigns a memory segment to store context association. Cursors are handles or pointers pointing to context regions. With cursors, stored procedures can control alterations in context regions.

NOTICE

If JDBC is used to call a stored procedure whose returned value is a cursor, the returned cursor cannot be used.

Cursors are classified into explicit cursors and implicit cursors. **Table 12-2** shows the usage conditions of explicit and implicit cursors for different SQL statements.

Table 12-2 Cursor usage conditions

SQL Statement	Cursor
Non-query statements	Implicit
Query statements with single-line results	Implicit or explicit
Query statements with multi-line results	Explicit

12.11.2 Explicit Cursor

An explicit cursor is used to process query statements, particularly when query results are multiple records.

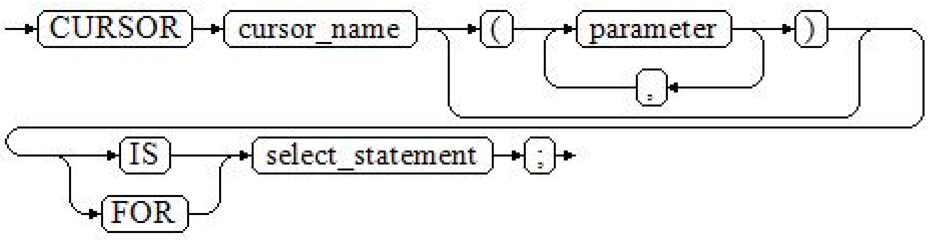
Procedure

An explicit cursor performs the following six PL/SQL steps to process query statements:

Step 1 Define a static cursor: Define a cursor name and its corresponding **SELECT** statement.

Figure 12-27 shows the syntax diagram for defining a static cursor.

Figure 12-27 static_cursor_define::=



Parameter description:

- *cursor_name*: defines a cursor name.
- *parameter*: specifies cursor parameters. Only input parameters are allowed. Its format is as follows:

```
parameter_name datatype
```

- *select_statement*: specifies a query statement.

NOTE

The system automatically determines whether the cursor can be used for backward fetching based on the execution plan.

Define a dynamic cursor: Define a **ref** cursor, which means that the cursor can be opened dynamically by a set of static SQL statements. Define the type of the **ref** cursor first, and then the cursor variable of this cursor type. Dynamically bind a **SELECT** statement through **OPEN FOR** when the cursor is opened.

Figure 12-28 and **Figure 12-29** show the syntax diagrams for defining a dynamic cursor.

Figure 12-28 cursor_typename::=

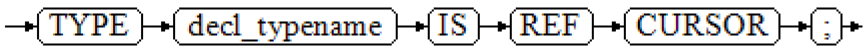
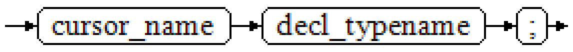


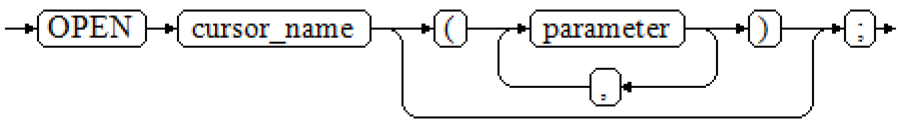
Figure 12-29 dynamic_cursor_define::=



Step 2 Open the static cursor: Execute the **SELECT** statement corresponding to the cursor. The query result is placed in the workspace and the pointer directs to the head of the workspace to identify the cursor result set. If the cursor query statement carries the **FOR UPDATE** option, the **OPEN** statement locks the data rows corresponding to the cursor result set in the database table.

Figure 12-30 shows the syntax diagram for opening a static cursor.

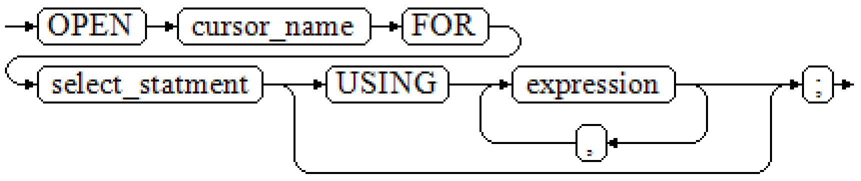
Figure 12-30 open_static_cursor::=



Open the dynamic cursor: Use the **OPEN FOR** statement to open the dynamic cursor and the SQL statement is dynamically bound.

Figure 12-31 shows the syntax diagrams for opening a dynamic cursor.

Figure 12-31 open_dynamic_cursor::=

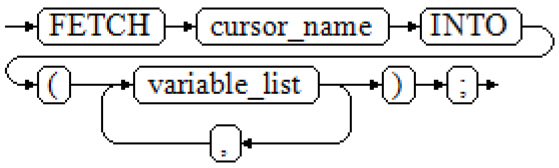


A PL/SQL program cannot use the **OPEN** statement to repeatedly open a cursor.

Step 3 Fetch cursor data: Retrieve data rows in the result set and place them in specified output variables.

Figure 12-32 shows the syntax diagrams for fetching cursor data.

Figure 12-32 fetch_cursor::=



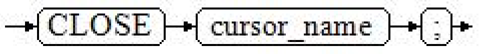
Step 4 Process the record.

Step 5 Continue to process until the active set has no record.

Step 6 Close the cursor: After you fetch and process the data in the cursor result set, close the cursor in time to release system resources used by the cursor and invalidate the workspace of the cursor so that the **FETCH** statement cannot be used to fetch data any more. A closed cursor can be reopened by an **OPEN** statement.

Figure 12-33 shows the syntax diagram for closing a cursor.

Figure 12-33 close_cursor::=



----End

Attributes

Cursor attributes are used to control program procedures or know program status. When a DML statement is executed, the PL/SQL opens a built-in cursor and processes its result. A cursor is a memory segment for maintaining query results. It is opened when a DML statement is executed and closed when the execution is finished. An explicit cursor has the following attributes:

- **%FOUND**: Boolean attribute, which returns **TRUE** if the last fetch returns a row.
- **%NOTFOUND**: Boolean attribute, which works opposite to the **%FOUND** attribute.
- **%ISOPEN**: Boolean attribute, which returns **TRUE** if the cursor has been opened.
- **%ROWCOUNT**: numeric attribute, which returns the number of records fetched from the cursor.

Examples

```
-- Specify the method for passing cursor parameters.
CREATE OR REPLACE PROCEDURE cursor_proc1()
AS
```

```
DECLARE
    DEPT_NAME VARCHAR(100);
    DEPT_LOC NUMBER(4);
    -- Define a cursor.
    CURSOR C1 IS
        SELECT section_name, place_id FROM hr.sections WHERE section_id <= 50;
    CURSOR C2(sect_id INTEGER) IS
        SELECT section_name, place_id FROM hr.sections WHERE section_id <= sect_id;
    TYPE CURSOR_TYPE IS REF CURSOR;
    C3 CURSOR_TYPE;
    SQL_STR VARCHAR(100);
BEGIN
    OPEN C1;-- Open the cursor.
    LOOP
        -- Fetch data from the cursor.
        FETCH C1 INTO DEPT_NAME, DEPT_LOC;
        EXIT WHEN C1%NOTFOUND;
        DBE_OUTPUT.PRINT_LINE(DEPT_NAME||'---'||DEPT_LOC);
    END LOOP;
    CLOSE C1;-- Close the cursor.

    OPEN C2(10);
    LOOP
        FETCH C2 INTO DEPT_NAME, DEPT_LOC;
        EXIT WHEN C2%NOTFOUND;
        DBE_OUTPUT.PRINT_LINE(DEPT_NAME||'---'||DEPT_LOC);
    END LOOP;
    CLOSE C2;

    SQL_STR := 'SELECT section_name, place_id FROM hr.sections WHERE section_id <= :DEPT_NO;';
    OPEN C3 FOR SQL_STR USING 50;
    LOOP
        FETCH C3 INTO DEPT_NAME, DEPT_LOC;
        EXIT WHEN C3%NOTFOUND;
        DBE_OUTPUT.PRINT_LINE(DEPT_NAME||'---'||DEPT_LOC);
    END LOOP;
    CLOSE C3;
END;
/

CALL cursor_proc1();

DROP PROCEDURE cursor_proc1;
-- Give a salary raise to employees whose salary is lower than 3000 by adding 500.
CREATE TABLE hr.staffs_t1 AS TABLE hr.staffs;

CREATE OR REPLACE PROCEDURE cursor_proc2()
AS
DECLARE
   V_EMPNO  NUMBER(6);
   V_SAL    NUMBER(8,2);
   CURSOR C IS SELECT staff_id, salary FROM hr.staffs_t1;
BEGIN
   OPEN C;
   LOOP
      FETCH C INTO V_EMPNO, V_SAL;
      EXIT WHEN C%NOTFOUND;
      IF V_SAL<=3000 THEN
            UPDATE hr.staffs_t1 SET salary =salary + 500 WHERE staff_id = V_EMPNO;
      END IF;
   END LOOP;
   CLOSE C;
END;
/

CALL cursor_proc2();

-- Delete the stored procedure.
```

```
DROP PROCEDURE cursor_proc2;
DROP TABLE hr.staffs_t1;
-- Use function parameters of the SYS_REFCURSOR type.
CREATE OR REPLACE PROCEDURE proc_sys_ref(O OUT SYS_REFCURSOR)
IS
C1 SYS_REFCURSOR;
BEGIN
OPEN C1 FOR SELECT section_ID FROM HR.sections ORDER BY section_ID;
O := C1;
END;
/

DECLARE
C1 SYS_REFCURSOR;
TEMP NUMBER(4);
BEGIN
proc_sys_ref(C1);
LOOP
  FETCH C1 INTO TEMP;
  DBE_OUTPUT.PRINT_LINE(C1%ROWCOUNT);
  EXIT WHEN C1%NOTFOUND;
END LOOP;
END;
/

-- Delete the stored procedure.
DROP PROCEDURE proc_sys_ref;
```

12.11.3 Implicit Cursor

Implicit cursors are automatically set by the system for non-query statements such as modify or delete operations, along with their workspace. Implicit cursors are named **SQL**, which is defined by the system.

Overview

Implicit cursor operations, such as definition, open, value-grant, and close operations, are automatically performed by the system and do not need users to process. Users can use only attributes related to implicit cursors to complete operations. In workspace of implicit cursors, the data of the latest SQL statement is stored and is not related to explicit cursors defined by users.

Format call: **SQL%**

NOTE

- **INSERT**, **UPDATE**, **DELETE**, and **SELECT** statements do not need defined cursors.
- In O-compatible mode, if the GUC parameter **behavior_compat_options** is set to **compat_cursor**, implicit cursors are valid across stored procedures.

Attributes

An implicit cursor has the following attributes:

- **SQL%FOUND**: Boolean attribute, which returns **TRUE** if the last fetch returns a row.
- **SQL%NOTFOUND**: Boolean attribute, which works opposite to the **SQL %FOUND** attribute.
- **SQL%ROWCOUNT**: numeric attribute, which returns the number of records fetched from the cursor.

- **SQL%ISOPEN**: Boolean attribute, whose value is always **FALSE**. Close implicit cursors immediately after an SQL statement is run.

Examples

```
-- Delete all employees in a department from the hr.staffs table. If the department has no employees, 
delete the department from the hr.sections table.
CREATE OR REPLACE PROCEDURE proc_cursor3()
AS
    DECLARE
    V_DEPTNO NUMBER(4) := 100;
    BEGIN
        DELETE FROM hr.staffs WHERE section_ID = V_DEPTNO;
        -- Proceed based on cursor status.
        IF SQL%NOTFOUND THEN
        DELETE FROM hr.sections_t1 WHERE section_ID = V_DEPTNO;
        END IF;
    END;
/

CALL proc_cursor3();

-- Delete the stored procedure and the temporary table.
DROP PROCEDURE proc_cursor3;
```

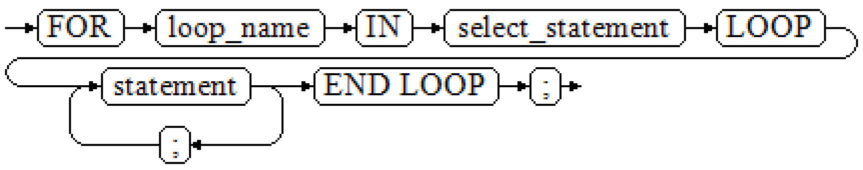
12.11.4 Cursor Loop

Use of cursors in **WHILE** and **LOOP** statements is called a cursor loop. Generally, **OPEN**, **FETCH**, and **CLOSE** statements are called in this kind of loop. The following describes a loop that simplifies a cursor loop without the need for these operations. This mode is applicable to a static cursor loop, without executing four steps about a static cursor.

Syntax

Figure 12-34 shows the syntax diagram of the **FOR AS** loop.

Figure 12-34 FOR_AS_loop::=



Precautions

- The **UPDATE** operation for the queried table is not allowed in the loop statement.
- The variable *loop_name* is automatically defined and is valid only in this loop. Its type is the same as that in the query result of *select_statement*. The value of *loop_name* is the query result of *select_statement*.
- The **%FOUND**, **%NOTFOUND**, and **%ROWCOUNT** attributes access the same internal variable in GaussDB. Transactions and the anonymous block do not support multiple cursor accesses at the same time.

Examples

```
BEGIN
FOR ROW_TRANS IN
        SELECT first_name FROM hr.staffs
    LOOP
        DBE_OUTPUT.PRINT_LINE (ROW_TRANS.first_name );
    END LOOP;
END;
/

-- Create a table.
CREATE TABLE integerTable1( A INTEGER);
CREATE TABLE integerTable2( B INTEGER);
INSERT INTO integerTable2 VALUES(2);

-- Multiple cursors share the parameters of cursor attributes.
DECLARE
   CURSOR C1 IS SELECT A FROM integerTable1;-- Declare the cursor.
   CURSOR C2 IS SELECT B FROM integerTable2;
   PI_A INTEGER;
   PI_B INTEGER;
BEGIN
    OPEN C1;-- Open the cursor.
   OPEN C2;
   FETCH C1 INTO PI_A; ---- The values of C1%FOUND and C2%FOUND are FALSE.
   FETCH C2 INTO PI_B; ---- The values of C1%FOUND and C2%FOUND are TRUE.
   -- Determine the cursor status.
   IF C1%FOUND THEN
       IF C2%FOUND THEN
         DBE_OUTPUT.PRINT_LINE('Dual cursor share paremeter.');
      END IF;
   END IF;
    CLOSE C1;-- Close the cursor.
   CLOSE C2;
END;
/

-- Delete the temporary table.
DROP TABLE integerTable1;
DROP TABLE integerTable2;
```

12.12 Retry Management

Retry is a process in which the database executes a SQL statement or stored procedure (including anonymous block) again in the case of execution failure, improving the execution success rate and user experience. The database checks the error code and retry configuration to determine whether to retry.

- If the execution fails, the system rolls back the executed statements and executes the stored procedure again.

  Example:

  ```
  openGauss=# CREATE OR REPLACE PROCEDURE retry_basic ( IN  x INT)
  AS
  BEGIN
        INSERT INTO t1 (a) VALUES (x);
        INSERT INTO t1 (a) VALUES (x+1);
  END;
  /

  openGauss=# CALL retry_basic(1);
  ```

12.13 Debugging

Syntax

RAISE

The syntax of RAISE is as follows:

Figure 12-35 raise_format::=

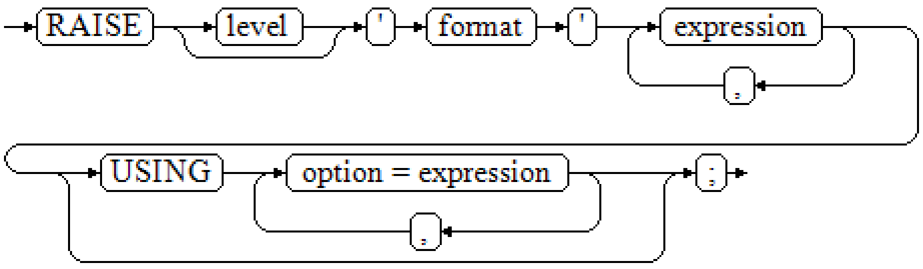


Figure 12-36 raise_condition::=

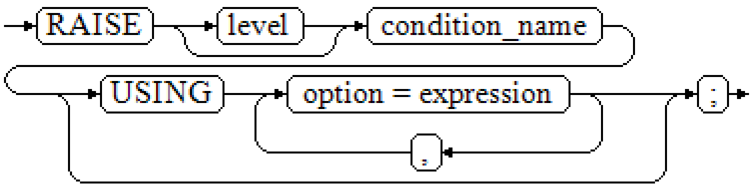


Figure 12-37 raise_sqlstate::=

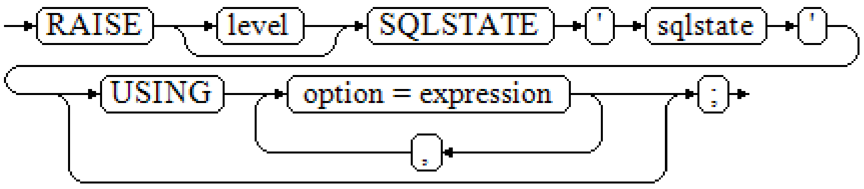


Figure 12-38 raise_option::=

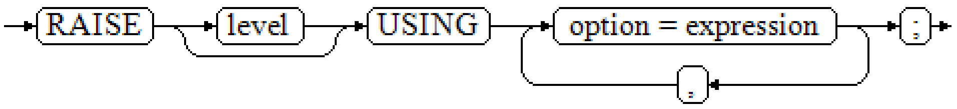
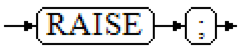


Figure 12-39 raise::=



Parameter description:

- The level option is used to specify the error level, that is, **DEBUG**, **LOG**, **INFO**, **NOTICE**, **WARNING**, or **EXCEPTION** (default). **EXCEPTION** throws an error that normally terminates the current transaction and the others only generate information at their levels. The log_min_messages and client_min_messages parameters control whether the error messages of specific levels are reported to the client and are written to the server log.
- **format**: specifies the error message text to be reported, a format string. The format string can be appended with an expression for insertion to the message text. In a format string, **%** is replaced by the parameter value attached to format and **%%** is used to print **%**. For example:

```
--v_job_id replaces % in the string.
RAISE NOTICE 'Calling cs_create_job(%)',v_job_id;
```

- **option = expression**: inserts additional information to an error report. The keyword option can be **MESSAGE**, **DETAIL**, **HINT**, or **ERRCODE**, and each expression can be any string.
  - **MESSAGE**: specifies the error message text. This option cannot be used in a **RAISE** statement that contains a format character string in front of **USING**.
  - **DETAIL**: specifies detailed information of an error.
  - **HINT**: prints hint information.
  - **ERRCODE**: designates an error code (SQLSTATE) to a report. A condition name or a five-character SQLSTATE error code can be used.
- **condition_name**: specifies the condition name corresponding to the error code.
- **sqlstate**: specifies the error code.

If neither a condition name nor an **SQLSTATE** is designated in a **RAISE EXCEPTION** command, the **RAISE EXCEPTION (P0001)** is used by default. If no message text is designated, the condition name or SQLSTATE is used as the message text by default.

NOTICE

- If the **SQLSTATE** designates an error code, the error code is not limited to a defined error code. It can be any error code containing five digits or ASCII uppercase rather than **00000**. Do not use an error code ended with three zeros because such error codes are category codes and can be captured by the whole category.
- In O-compatible mode, SQLCODE is equivalent to SQLSTATE.

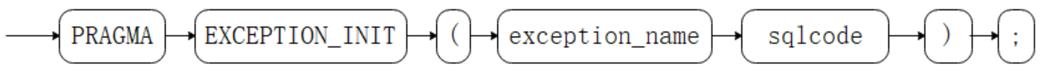
NOTE

The syntax described in **Figure 12-39** does not append any parameter. This form is used only for the **EXCEPTION** statement in a **BEGIN** block so that the error can be re-processed.

EXCEPTION_INIT

In O-compatible mode, EXCEPTION_INIT can be used to define the SQLCODE error code. The syntax is as follows:

Figure 12-40 exception_init::=



Parameter description:

- **exception_name** indicates the name of the exception declared by the user. The **EXCEPTION_INIT** syntax must follow the declared exception.
- **sqlcode** is a customized SQL code, which must be a negative integer ranging from –2147483647 to –1.

NOTICE

When **EXCEPTION_INIT** is used to customize an SQL code, SQLSTATE is equivalent to SQLCODE, and SQLERRM is in the format of *xxx*: **non-GaussDB Exception**. For example, if the customized SQL code is **–1**, SQLSTATE is **–1** and SQLERRM is **1: non-GaussDB Exception**.

Example

Display error and hint information when a transaction terminates:

```
CREATE OR REPLACE PROCEDURE proc_raise1(user_id in integer)
AS
BEGIN
RAISE EXCEPTION 'Noexistence ID --> %',user_id USING HINT = 'Please check your user ID';
END;
/

call proc_raise1(300011);

-- Execution result:
ERROR:  Noexistence ID --> 300011
HINT:  Please check your user ID
```

Two methods are available for setting **SQLSTATE**:

```
CREATE OR REPLACE PROCEDURE proc_raise2(user_id in integer)
AS
BEGIN
RAISE 'Duplicate user ID: %',user_id USING ERRCODE = 'unique_violation';
END;
/

\set VERBOSITY verbose
call proc_raise2(300011);

-- Execution result:
ERROR:  Duplicate user ID: 300011
SQLSTATE: 23505
LOCATION:  exec_stmt_raise, pl_exec.cpp:3482
```

If the main parameter is a condition name or **SQLSTATE**, the following applies:

RAISE division_by_zero;

RAISE SQLSTATE '22012';

For example:

```
CREATE OR REPLACE PROCEDURE division(div in integer, dividend in integer)
AS
DECLARE
res int;
    BEGIN
    IF dividend=0 THEN
        RAISE division_by_zero;
        RETURN;
    ELSE
        res := div/dividend;
        RAISE INFO 'division result: %', res;
        RETURN;
    END IF;
    END;
/
call division(3,0);

-- Execution result:
ERROR:  division_by_zero
```

Alternatively:

```
RAISE unique_violation USING MESSAGE = 'Duplicate user ID: ' || user_id;
```

In O-compatible mode, EXCEPTION_INIT can be used to customize error codes SQLCODE.

```
declare
    deadlock_detected exception;
    pragma exception_init(deadlock_detected, -1);
begin
    if 1 > 0 then
        raise deadlock_detected;
    end if;
exception
    when deadlock_detected then
        raise notice 'sqlcode:%,sqlstate:%,sqlerrm:%',sqlcode,sqlstate,sqlerrm;
end;
/
-- Execution result:
NOTICE:  sqlcode:-1,sqlstate:-1,sqlerrm: 1: non-GaussDB Exception
```

12.14 Package

A package is a combination of PL/SQL programs, such as stored procedures, functions, variables, constants, and cursors. It is object-oriented and can encapsulate PL/SQL program design elements. Functions in a package are created, deleted, and modified in a unified manner.

A package contains two parts: package specifications and package body. The declaration contained in the package specifications can be accessed by external functions and anonymous blocks. The declaration contained in the package body cannot be accessed by external functions or anonymous blocks, but can be accessed only by functions and stored procedures in the package body.

For details about how to create a package, see **CREATE PACKAGE**.

NOTICE

- Cross-package variables cannot be used as control variables in the for loops.
- Types defined in a package cannot be deleted or modified, and cannot be used to define tables.
- Cursor variables cannot be referenced in SCHEMA.PACKAGE.CUROSR mode.
- A cursor with parameters can be opened only in the current package.

13 System Catalogs and System Views

13.1 Overview of System Catalogs and System Views

System catalogs store the structured metadata of GaussDB. They are the source of information used by GaussDB to control system running and are a core component of the database system.

System views provide ways to query the system catalogs and internal database status.

System catalogs and system views are visible to either system administrators or all users. Some system catalogs and views have marked the need of administrator permissions, so they are accessible only to administrators.

You can delete and re-create system catalogs, add columns to them, and insert and update values in them, but doing so may make system information inconsistent and cause system faults. Generally, you should not modify system catalogs or system views, or rename their schemas. They are automatically maintained by the system.

NOTICE

- You are not advised to modify the permissions on system catalogs or system views.
- Do not add, delete, or modify system catalogs because doing so will result in exceptions or even database unavailability.
- For details about field types in system catalogs and system views, see section **Data Types**.

13.2 System Catalogs

13.2.1 GS_ASP

GS_ASP displays the persistent ACTIVE SESSION PROFILE samples. This catalog can be queried only in the system database but cannot be queried in the user database.

Table 13-1 GS_ASP columns

Name	Type	Description
sampleid	bigint	Sample ID
sample_time	timestamp with time zone	Sampling time
need_flush_sample	boolean	Specifies whether the sample needs to be flushed to disks. ● **t** (true): yes ● **f** (false): no
databaseid	oid	Database ID
thread_id	bigint	Thread ID
sessionid	bigint	Session ID
start_time	timestamp with time zone	Start time of a session
event	text	Event name
lwtid	integer	Lightweight thread ID of the current thread
psessionid	bigint	Parent thread of the streaming thread
tlevel	integer	Level of the streaming thread, which corresponds to the level (**id**) of the execution plan
smpid	integer	Concurrent thread ID in SMP execution mode
userid	oid	ID of a session user
application_name	text	Name of an application
client_addr	inet	IP address of a client
client_hostname	text	Name of a client
client_port	integer	TCP port number used by a client to communicate with the backend
query_id	bigint	Debug query ID

Name	Type	Description
unique_query_id	bigint	Unique query ID
user_id	oid	User ID in the key of the unique query
cn_id	integer	ID of the node that delivers the unique SQL statement. **cn_id** is in the key of the unique query.
unique_query	text	Standardized UniqueSQL text string
locktag	text	Information of a lock that the session waits for, which can be parsed using **locktag_decode**
lockmode	text	Mode of a lock that the session waits for ● **LW_EXCLUSIVE**: exclusive lock ● **LW_SHARED**: shared lock ● **LW_WAIT_UNTIL_FREE**: waits for the **LW_EXCLUSIVE** to be available
block_sessionid	bigint	Blocks a session from obtaining the session ID of a lock if the session is waiting for the lock.
wait_status	text	Provides more details about an event column.
global_sessionid	text	Global session ID
xact_start_time	timestamp with time zone	Start time of the transaction
query_start_time	timestamp with time zone	Time when the statement starts to be executed
state	text	Current transaction state The value can be **active**, **idle in transaction**, **fastpath function call**, **idle in transaction (aborted)**, **disabled**, or **retrying**.

13.2.2 GS_AUDITING_POLICY

GS_AUDITING_POLICY records the main information about the unified audit. Each record corresponds to a design policy. Only the system administrator or security policy administrator can access this system catalog.

Table 13-2 GS_AUDITING_POLICY columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified)
polname	name	Policy name, which must be unique
polcomments	name	Policy description field, which records policy-related description information and is represented by the **COMMENTS** keyword
modifydate	timestamp without time zone	Latest timestamp when a policy is created or modified
polenabled	boolean	Specifies whether to enable a policy.

13.2.3 GS_AUDITING_POLICY_ACCESS

GS_AUDITING_POLICY_ACCESS records the unified audit information about DML database operations. Only the system administrator or security policy administrator can access this system catalog.

Table 13-3 GS_AUDITING_POLICY_ACCESS columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified)
accesstype	name	DML database operation type. For example, SELECT, INSERT, and DELETE.
labelname	name	Resource label name. This column corresponds to the **polname** column in the **GS_AUDITING_POLICY** system catalog.
policyoid	oid	OID of the audit policy, corresponding to the OID in the **GS_AUDITING_POLICY** system catalog
modifydate	timestamp without time zone	Latest creation or modification timestamp

13.2.4 GS_AUDITING_POLICY_FILTERS

GS_AUDITING_POLICY_FILTERS records the filtering policies about the unified audit. Each record corresponds to a design policy. Only the system administrator or security policy administrator can access this system catalog.

Table 13-4 GS_AUDITING_POLICY_FILTERS columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
filtertype	name	Filter type. Currently, the value is **logical_expr**.
labelname	name	Name. Currently, the value is **logical_expr**.
policyoid	oid	OID of the audit policy, corresponding to the OID in the **GS_AUDITING_POLICY** system catalog
modifydate	timestamp without time zone	Latest creation or modification timestamp
logicaloperator	text	Logical character string of a filter criterion

13.2.5 GS_AUDITING_POLICY_PRIVILEGES

GS_AUDITING_POLICY_PRIVILEGES records the DDL database operations about the unified audit. Each record corresponds to a design policy. Only the system administrator or security policy administrator can access this system catalog.

Table 13-5 GS_AUDITING_POLICY_PRIVI columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
privilegetype	name	DDL database operation type. For example, CREATE, ALTER, and DROP.
labelname	name	Resource label name. This column corresponds to the **polname** column in the **GS_AUDITING_POLICY** system catalog.

Name	Type	Description
policyoid	oid	This column corresponds to the OID in the **GS_AUDITING_POLICY** system catalog.
modifydate	timestamp without time zone	Latest creation or modification timestamp

13.2.6 GS_CLIENT_GLOBAL_KEYS

GS_CLIENT_GLOBAL_KEYS records information about the CMK in the encrypted equality feature. Each record corresponds to a CMK.

Table 13-6 GS_CLIENT_GLOBAL_KEYS columns

Name	Type	Description
oid	oid	Row identifier (hidden column)
global_key_name	name	CMK name
key_namespace	oid	Namespace OID that contains the CMK
key_owner	oid	CMK owner
key_acl	aclitem[]	Access permissions that the key should have on creation
create_date	timestamp without time zone	Time when the key is created

13.2.7 GS_CLIENT_GLOBAL_KEYS_ARGS

GS_CLIENT_GLOBAL_KEYS_ARGS records the metadata about the CMK in the encrypted equality feature. Each record corresponds to a key-value pair of the CMK.

Table 13-7 GS_CLIENT_GLOBAL_KEYS_ARGS columns

Name	Type	Description
oid	oid	Row identifier (hidden column)
global_key_id	oid	CMK OID
function_name	name	The value is **encryption**.
key	name	CMK metadata name

Name	Type	Description
value	bytea	Value of the CMK metadata name

13.2.8 GS_COLUMN_KEYS

GS_COLUMN_KEYS records information about the CEK in the encrypted equality feature. Each record corresponds to a CEK.

Table 13-8 GS_COLUMN_KEYS columns

Name	Type	Description
oid	oid	Row identifier (hidden column)
column_key_name	name	CEK name
column_key_distributed_id	oid	ID obtained based on the hash value of the fully qualified domain name (FQDN) of the CEK
global_key_id	oid	Foreign key, which is the CMK OID
key_namespace	oid	Namespace OID that contains the CEK
key_owner	oid	CEK owner
create_date	timestamp without time zone	Time when the CEK is created
key_acl	aclitem[]	Access permissions that the CEK should have on creation

13.2.9 GS_COLUMN_KEYS_ARGS

GS_COLUMN_KEYS_ARGS records the metadata about the CMK in the encrypted equality feature. Each record corresponds to a key-value pair of the CMK.

Table 13-9 GS_COLUMN_KEYS_ARGS columns

Name	Type	Description
oid	oid	Row identifier (hidden column)
column_key_id	oid	CEK OID
function_name	name	The value is **encryption**.
key	name	CEK metadata name
value	bytea	Value of the CEK metadata name

13.2.10 GS_DB_PRIVILEGE

GS_DB_PRIVILEGE records the granting of ANY permissions. Each record corresponds to a piece of authorization information.

Table 13-10 GS_DB_PRIVILEGE columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
roleid	oid	User ID.
privilege_type	text	ANY permission of a user. For details about the value, see **Table 11-122**.
admin_option	boolean	Whether the ANY permission recorded in the **privilege_type** column can be re-granted. ● **t**: yes. ● **f**: no.

13.2.11 GS_ENCRYPTED_COLUMNS

GS_ENCRYPTED_COLUMNS records information about encrypted columns in the encrypted equality feature. Each record corresponds to an encrypted column.

Table 13-11 GS_ENCRYPTED_COLUMNS columns

Name	Type	Description
oid	oid	Row identifier (hidden column)
rel_id	oid	Table OID
column_name	name	Name of an encrypted column.
column_key_id	oid	Foreign key, which is the CEK OID
encryption_type	tinyint	Encryption type. The value can be **2 (DETERMINISTIC)** or **1 (RANDOMIZED)**.
data_type_original_oid	oid	ID of the original data type of the encrypted column. For details about the value, see **oid** in the **PG_TYPE** system catalog.

Name	Type	Description
data_type_original_mod	integer	Original data type modifier of the encrypted column. For details about the value, see **atttypmod** in the PG_ATTRIBUTE system catalog. The value of **data_type_original_mod** is generally **–1** when data types are not specific.
create_date	timestamp without time zone	Time when an encrypted column is created

13.2.12 GS_ENCRYPTED_PROC

GS_ENCRYPTED_PROC provides information such as the parameters of encrypted functions and stored procedure functions, original data type of return values, and encrypted columns.

Table 13-12 GS_ENCRYPTED_PROC columns

Name	Type	Description
oid	oid	Row identifier (hidden column)
func_id	oid	OID of the function, corresponding to the OID row identifier in the **pg_proc** system catalog.
prorettype_orig	integer	Original data type of the return value
last_change	timestamp without time zone	Last modification time of the encrypted function information
proargcachedcol	oidvector	OID of the encrypted column corresponding to the **INPUT** parameter of the function, corresponding to the OID row identifier in the **gs_encrypted_columns** system catalog
proallargtypes_orig	oid[]	Original data type of all function parameters

13.2.13 GS_GLOBAL_CHAIN

GS_GLOBAL_CHAIN records information about modification operations performed by users on the tamper-proof user table. Each record corresponds to a table-level modification operation. Users with the audit administrator permission can query this system catalog, but no user is allowed to modify this system catalog.

Table 13-13 GS_GLOBAL_CHAIN columns

Name	Type	Description
blocknum	bigint	Block number, which is the sequence number of the current user operation recorded in the ledger
dbname	name	Name of the database, to which the modified tamper-proof user table belongs
username	name	Username, which is the name of the user who performs the operation of modifying the user table
starttime	timestamp with time zone	Latest timestamp when a user performs an operation
relid	oid	OID of the modified tamper-proof user table
relnsp	name	OID of the namespace to which the modified tamper-proof user table belongs
relname	name	User table name, which is the name of the modified tamper-proof user table
relhash	hash16	Table-level hash change amount generated by the current operation
globalhash	hash32	Global digest, which is calculated based on the information of the current row and the **globalhash** of the previous row. It connects the entire table to verify the integrity of **GS_GLOBAL_CHAIN** data.
txcommand	text	SQL statement whose operations are recorded

13.2.14 GS_GLOBAL_CONFIG

GS_GLOBAL_CONFIG records the parameter values specified by users during database instance initialization. In addition, it also stores weak passwords set by users. Initial database users can write, modify, and delete parameters in system catalogs using ALTER and DROP.

Table 13-14 GS_GLOBAL_CONFIG columns

Name	Type	Description
name	name	Specifies the preset parameter name, weak password name, or parameter required by users during database instance initialization.

Name	Type	Description
value	text	Specifies the preset parameter value, weak password value, or parameter value required by users during database instance initialization.

13.2.15 GS_JOB_ARGUMENT

GS_JOB_ARGUMENT provides the parameter attributes of DBE_SCHEDULER scheduled tasks and programs.

Table 13-15 GS_JOB_ARGUMENT columns

Name	Type	Description
oid	oid	Row identifier (hidden column)
argument_position	integer	Location of a parameter of a scheduled task or program.
argument_type	name	Parameter type of a scheduled task or program.
job_name	text	Name of a scheduled task or program.
argument_name	text	Parameter name of a scheduled task or program. The scheduled task inherits the parameter name of the program. Therefore, this parameter is null.
argument_value	text	Parameter value of a scheduled task. (The program cannot bind a value.)
default_value	text	Default parameter value of a program.

13.2.16 GS_JOB_ATTRIBUTE

GS_JOB_ATTRIBUTE records attributes of DBE_SCHEDULER scheduled tasks, including basic attributes of scheduled tasks, scheduled task classes, certificates, authorization, programs, and schedules.

Table 13-16 GS_JOB_ATTRIBUTE columns

Name	Type	Description
oid	oid	Row identifier (hidden column)

Name	Type	Description
job_name	text	Names of scheduled tasks, scheduled task classes, certificates, programs, and schedules, and authorized user names.
attribute_name	text	Attribute names of scheduled tasks, scheduled task classes, certificates, programs, and schedules, and authorized content.
attribute_value	text	Attribute values of scheduled tasks, scheduled task classes, certificates, programs, and schedules.

13.2.17 GS_MASKING_POLICY

GS_MASKING_POLICY records the main information about dynamic data masking policies. Each record corresponds to a masking policy. Only the system administrator or security policy administrator can access this system catalog.

Table 13-17 GS_MASKING_POLICY columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
polname	name	Policy name, which must be unique
polcomments	name	Policy description field, which records policy-related description information and is represented by the **COMMENTS** keyword
modifydate	timestamp without time zone	Latest timestamp when a policy is created or modified
polenabled	boolean	Specifies whether to enable a policy.

13.2.18 GS_MASKING_POLICY_ACTIONS

GS_MASKING_POLICY_ACTIONS records the masking actions of a masking policy in the dynamic data masking policies. One masking policy corresponds to one or more rows of records in the catalog. Only the system administrator or security policy administrator can access this system catalog.

Table 13-18 GS_MASKING_POLICY_ACTIONS columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
actiontype	name	Name of a masking function used by a masking policy
actparams	name	Parameter information transferred to a masking function
actlabelname	name	Name of a masked label
policyoid	oid	OID of a masking policy to which a record belongs, corresponding to the OID in **GS_MASKING_POLICY**
actmodifydate	timestamp without time zone	Latest timestamp when a record is created or modified

13.2.19 GS_MASKING_POLICY_FILTERS

GS_MASKING_POLICY_FILTERS records the user filter criteria corresponding to the dynamic data masking policies. The corresponding masking policy takes effect only when the user information meets the filter criteria. Only the system administrator or security policy administrator can access this system catalog.

Table 13-19 GS_MASKING_POLICY_FILTERS columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
filtertype	name	Filter type. Currently, the value is **logical_expr**.
filterlabelname	name	Filter range. Currently, the value is **logical_expr**.
policyoid	oid	OID of the masking policy to which a user filter criterion belongs, which corresponds to the OID in **GS_MASKING_POLICY**
modifydate	timestamp without time zone	Latest timestamp when a user filter criterion is created or modified
logicaloperator	text	Polish notation of a filter criterion

13.2.20 GS_MATVIEW

GS_MATVIEW provides information about each materialized view in the database.

Table 13-20 GS_MATVIEW columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified)
matviewid	oid	OID of a materialized view
mapid	oid	OID of a map table associated with a materialized view. Each map table corresponds to one materialized view. If a complete-refresh materialized view does not correspond to a map table, the value of this column is **0**.
ivm	boolean	Type of a materialized view. The value **t** indicates a fast-refresh materialized view, and the value **f** indicates a complete-refresh materialized view.
needrefresh	boolean	Reserved column
refreshtime	timestamp	Last time when a materialized view was refreshed. If the materialized view is not refreshed, the value is null. This column is maintained only for fast-refresh materialized views. For complete-refresh materialized views, the value is null.

13.2.21 GS_MATVIEW_DEPENDENCY

GS_MATVIEW_DEPENDENCY provides association information about each fast-refresh materialized view, base table, and Mlog table in the database. The Mlog table corresponding to the base table does not exist in the complete-refresh materialized view. Therefore, no record is written into the Mlog table.

Table 13-21 GS_MATVIEW_DEPENDENCY columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified)
matviewid	oid	OID of a materialized view
relid	oid	OID of a base table of a materialized view

Name	Type	Description
mlogid	oid	OID of a Mlog table which is the log table of a materialized view. Each Mlog table corresponds to one base table.
mxmin	int4	Reserved column

13.2.22 GS_MODEL_WAREHOUSE

GS_MODEL_WAREHOUSE stores AI engine training models, including the models and detailed description of the training process.

Table 13-22 GS_MODEL_WAREHOUSE columns

Name	Data Type	Description
modelname	name	Unique constraint
modelowner	oid	OID of a model owner
createtime	timestamp without time zone	Time when a model is created
processedtuples	integer	Number of tuples involved in training
discardedtuples	integer	Number of unqualified tuples not involved in training
preprocesstime	real	Data preprocessing time
exectime	real	Training duration
iterations	integer	Iteration round
outputtype	oid	OID of the output data type
modeltype	text	AI operator type
query	text	Query statement executed to create a model
modeldata	bytea	Stored binary model information
weight	real[]	Currently, this column applies only to GD operator models.

Name	Data Type	Description
hyperparametersnames	text[]	Involved hyperparameter name
hyperparametersvalues	text[]	Hyperparameter value
hyperparametersoids	oid[]	OID of the data type corresponding to a hyperparameter
coefnames	text[]	Model parameter
coefvalues	text[]	Value of a model parameter
coefoids	oid[]	OID of the data type corresponding to a model parameter
trainingscoresname	text[]	Method used to measure model performance
trainingscoresvalue	real[]	Value used to measure model performance
modeldescribe	text[]	Model description

13.2.23 GS_OPT_MODEL

GS_OPT_MODEL is a data table used when the AI engine is enabled to predict the planned time. It records the configurations, training results, features, corresponding system functions, and training history of machine learning models.

Table 13-23 GS_OPT_MODEL columns

Name	Type	Description
oid	oid	Database object ID
template_name	name	Template name of the machine learning model, which determines the interfaces invoked for training and prediction. Currently, only rlstm is implemented.

Name	Type	Description
model_name	name	Model name. Each model corresponds to a set of parameters, training logs, and model coefficients in the AI engine online learning process. The name must be unique.
datname	name	Name of the database served by the model. Each model is specific to a single database. This parameter determines data used for training.
ip	name	IP address of the host where the AI engine is deployed
port	integer	Listening port number of the AI engine
max_epoch	integer	Maximum number of iterations in an epoch
learning_rate	real	Learning rate of model training. The default value **1** is recommended.
dim_red	real	Number of model feature dimensions whose retention is reduced
hidden_units	integer	Number of neurons in the model's hidden layer. If the model cannot be converged for a long time, increase the value of this parameter.
batch_size	integer	Size of a batch in each iteration. It is recommended that the size be greater than or equal to the total training data volume to accelerate model convergence.

Name	Type	Description
feature_size	integer	Length of the model feature, which is used to trigger retraining. This parameter is automatically updated after model training and does not need to be specified.
available	boolean	Whether the model is converged. This parameter does not need to be specified.
Is_training	boolean	Whether the model is being trained. This parameter does not need to be specified.
label	"char"[]	Target task of the model. ● S: startup time ● T: total time ● R: rows ● M: peak memory Currently, {S, T} or {R} is recommended due to model performance restrictions.
max	bigint[]	Maximum value of each task label of the model, which is used to trigger retraining. This parameter does not need to be specified.
acc	real[]	Accuracy of each model task. This parameter does not need to be specified.
description	text	Model comment

13.2.24 GS_PACKAGE

GS_PACKAGE records package information.

Table 13-24 GS_PACKAGE columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified)
pkgnamespace	oid	Schema to which a package belongs
pkgowner	oid	Owner of a package
pkgname	name	Name of a package
pkgspecsrc	text	Package specification
pkgbodydeclsrc	text	Package body
pkgbodyinitsrc	text	Package initialization source
pkgacl	aclitem[]	Access permission
pkgsecdef	boolean	Whether a user has the definer permission on the package.

13.2.25 GS_POLICY_LABEL

GS_POLICY_LABEL records the resource label configuration information. One resource label corresponds to one or more records, and each record identifies the resource label to which a database resource belongs. Only the system administrator or security policy administrator can access this system catalog.

Fully Qualified Domain Name (FQDN) identifies an absolute path of a database resource.

Table 13-25 GS_POLICY_LABEL columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified)
labelname	name	Resource label name
labeltype	name	Resource tag type. Currently, the value is **RESOURCE**.
fqdnnamespace	oid	OID of a namespace to which an identified database resource belongs
fqdnid	oid	OID of an identified database resource. If the database resource is a column, this column is the OID of the catalog.

Name	Type	Description
relcolumn	name	Column name. If the identified database resource is a column, this column indicates the column name. Otherwise, this column is empty.
fqdntype	name	Type of the identified database resource, for example, schema, table, column, or view

13.2.26 GS_RECYCLEBIN

gs_recyclebin describes details about objects in the recycle bin.

Table 13-26 gs_recyclebin columns

Name	Type	Description
oid	oid	System column
rcybaseid	oid	Base table object ID, which references **gs_recyclebin.oid**
rcydbid	oid	OID of the database to which the current object belongs
rcyrelid	oid	OID of the current object
rcyname	name	Name of the object in the recycle bin. The format is BIN*$unique_id$oid$*0. *unique_id* indicates the unique identifier with a maximum of 16 characters, and *oid* indicates the OID.
rcyoriginname	name	Original object name
rcyoperation	"char"	Operation type ● **d**: drop ● **t**: truncate

Name	Type	Description
rcytype	integer	Object type ● **0**: table ● **1**: index ● **2**: TOAST table ● **3**: TOAST index ● **4**: sequence, indicating the sequence object that is automatically associated with the serial, bigserial, smallserial, and largeserial types. ● **5**: partition. ● **6**: global index.
rcyrecyclecsn	bigint	CSN when an object is dropped or truncated
rcyrecycletime	timestamp with time zone	Time when an object is dropped or truncated
rcycreatecsn	bigint	CSN when an object is created
rcychangecsn	bigint	CSN when an object definition is modified
rcynamespace	oid	OID of the namespace that contains this relationship
rcyowner	oid	Relationship owner
rcytablespace	oid	Tablespace where the relationship is stored. If the value is **0**, the default tablespace of the database is used. This column is meaningless if the relationship has no on-disk file.
rcyrelfilenode	oid	File name of the recycle bin object on a disk, or **0** if none, used to restore the texture file when the **TRUNCATE** object is restored.
rcycanrestore	bool	Whether flashback can be performed separately
rcycanpurge	bool	Whether the purge operation can be performed independently
rcyfrozenxid	xid32	All transaction IDs before this one have been replaced with a permanent (frozen) transaction ID in the table.
rcyfrozenxid64	xid	All transaction IDs before this one have been replaced with a permanent (frozen) transaction ID in the table.

13.2.27 GS_SQL_PATCH

GS_SQL_PATCH records the status information about all SQL patches.

Table 13-27 GS_SQL_PATCH columns

Name	Type	Description
patch_name	name	Patch name.
unique_sql_id	bigint	Global unique ID.
owner	oid	ID of the user who creates the patch.
enable	bool	Specifies whether the patch takes effect.
status	"char"	Patch status (reserved column).
abort	bool	Specifies whether the patch is an abort hint.
hint_string	text	Hint text.
hint_node	pg_node_tree	Hint parsing and serialization result.
original_query	text	Original statement (reserved column).
patched_query	text	Patched statement (reserved column).
original_query_tree	pg_node_tree	Original statement parsing result (reserved column).
patched_query_tree	pg_node_tree	Patched statement parsing result (reserved column).
description	text	Patch description.

13.2.28 GS_TXN_SNAPSHOT

GS_TXN_SNAPSHOT is a timestamp-CSN mapping table. It periodically samples and maintains an appropriate time range to estimate the CSN value corresponding to the timestamp in the range.

Table 13-28 GS_TXN_SNAPSHOT columns

Name	Data Type	Description
snptime	timestamp with time zone	Snapshot time
snpxmin	bigint	Minimum transaction ID snapshots

Name	Data Type	Description
snpcsn	bigint	Commit sequence number (CSN) snapshots
snpsnapshot	text	Serialized snapshot text

13.2.29 GS_UID

GS_UID records the unique identification meta information of the hasuids attribute table in the database.

Table 13-29 GS_UID columns

Name	Type	Description
relid	oid	OID of a table.
uid_backup	bigint	Largest unique identifier that can be assigned to a table.

13.2.30 GS_WLM_EC_OPERATOR_INFO

GS_WLM_EC_OPERATOR_INFO records operator information after an Extension Connector job ends. If **enable_resource_record** is set to **on**, the system imports records from **GS_WLM_EC_OPERATOR_HISTORY** to this system catalog every 3 minutes. This operation occupies storage space and affects performance. The sysadmin permission is required for querying the system catalog. The current feature is a lab feature. Contact Huawei technical support before using it.

Table 13-30 GS_WLM_EC_OPERATOR_INFO columns

Name	Type	Description
queryid	bigint	Internal query ID used for Extension Connector statement execution
plan_node_id	integer	Plan node ID of the execution plan of an Extension Connector operator
start_time	timestamp with time zone	Time when the Extension Connector operator starts to process the first data record
duration	bigint	Total execution time of the Extension Connector operator (unit: ms)
tuple_processed	bigint	Number of elements returned by the Extension Connector operator

Name	Type	Description
min_peak_memory	integer	Minimum peak memory used by the Extension Connector operator on all DNs (unit: MB)
max_peak_memory	integer	Maximum peak memory used by the Extension Connector operator on all DNs (unit: MB)
average_peak_memory	integer	Average peak memory used by the Extension Connector operator on all DNs (unit: MB)
ec_status	text	Status of the Extension Connector job
ec_execute_datanode	text	Name of the DN executing an Extension Connector job
ec_dsn	text	DSN used by an Extension Connector job
ec_username	text	Username used by the Extension Connector job to access a remote database instance (the value is null if the remote database instance is SPARK.)
ec_query	text	Statement sent by an Extension Connector job to a remote database instance
ec_libodbc_type	text	Type of the unixODBC driver used by an Extension Connector job

13.2.31 GS_WLM_INSTANCE_HISTORY

GS_WLM_INSTANCE_HISTORY records information about resource usage related to instances (primary database node or database node). Each record in this system catalog indicates resource usage of an instance at a specific time point, including the memory, number of CPU cores, disk I/O, physical I/O of the process, and logical I/O of the process. This system catalog can be queried by users with the **sysadmin** permission only in Postgres.

Table 13-31 GS_WLM_INSTANCE_HISTORY columns

Name	Type	Description
instancename	text	Instance name.
timestamp	timestamp with time zone	Timestamp.
used_cpu	integer	CPU usage of the instance.
free_mem	integer	Unused memory of the instance, in MB
used_mem	integer	Used memory of the instance, in MB

Name	Type	Description
io_await	real	Average wait time for an I/O operation on the disk used by the instance. The average value is within 10 seconds.
io_util	real	io_util value of the disk used by the instance. The average value is within 10 seconds.
disk_read	real	Disk read rate of the instance, in KB/s. The average value is within 10 seconds.
disk_write	real	Disk write rate of the instance, in KB/s. The average value is within 10 seconds.
process_read	bigint	Read rate (excluding the number of bytes read from the disk pagecache) of the corresponding instance process that reads data from a disk within 10 seconds, in KB/s
process_write	bigint	Write rate (excluding the number of bytes written to the disk pagecache) of the corresponding instance process that writes data to a disk within 10 seconds, in KB/s
logical_read	bigint	Primary database node instance: not collected Database node instance: logical read byte rate of the instance within the statistical interval (10 seconds), in KB/s
logical_write	bigint	Primary database node instance: not collected Database node instance: logical write byte rate of the instance within the statistical interval (10 seconds), in KB/s
read_counts	bigint	master database node instance: Statistics are not collected. Database node instance: total number of logical read operations of the instance within the statistical interval (10 seconds)
write_counts	bigint	master database node instance: Statistics are not collected. Database node instance: total number of logical write operations of the instance within the statistical interval (10 seconds)

13.2.32 GS_WLM_OPERATOR_INFO

GS_WLM_OPERATOR_INFO displays records about operators of completed jobs. The data is dumped from the kernel to the system catalog. This system catalog can be queried by users with the **sysadmin** permission only in Postgres.

Table 13-32 GS_WLM_OPERATOR_INFO columns

Name	Type	Description
queryid	bigint	Internal query ID used for statement execution
pid	bigint	Thread ID of the backend
plan_node_id	integer	Plan node ID of the execution plan of a query
plan_node_name	text	Name of the operator corresponding to the plan node ID
start_time	timestamp with time zone	Time when the operator starts to process the first data record
duration	bigint	Total execution time of the operator, in ms
query_dop	integer	DOP of the operator
estimated_rows	bigint	Number of rows estimated by the optimizer
tuple_processed	bigint	Number of elements returned by the operator
min_peak_memory	integer	Minimum peak memory used by the operator on database nodes (unit: MB)
max_peak_memory	integer	Maximum peak memory used by the operator on database nodes (unit: MB)
average_peak_memory	integer	Average peak memory used by the operator on database nodes (unit: MB)
memory_skew_percent	integer	Memory usage skew of the operator among database nodes
min_spill_size	integer	Minimum spilled data among database nodes when a spill occurs (unit: MB) (default value: **0**)
max_spill_size	integer	Maximum spilled data among database nodes when a spill occurs (unit: MB) (default value: **0**)
average_spill_size	integer	Average spilled data among database nodes when a spill occurs (unit: MB) (default value: **0**)
spill_skew_percent	integer	Database node spill skew when a spill occurs
min_cpu_time	bigint	Minimum execution time of the operator on database nodes (unit: ms)
max_cpu_time	bigint	Maximum execution time of the operator on database nodes (unit: ms)

Name	Type	Description
total_cpu_time	bigint	Total execution time of the operator on database nodes (unit: ms)
cpu_skew_percent	integer	Execution time skew among database nodes
warning	text	Warning. The following warnings are displayed: ● Sort/SetOp/HashAgg/HashJoin spill ● Spill file size large than 256MB ● Broadcast size large than 100MB ● Early spill ● Spill times is greater than 3 ● Spill on memory adaptive ● Hash table conflict

13.2.33 GS_WLM_PLAN_ENCODING_TABLE

GS_WLM_PLAN_ENCODING_TABLE displays encoding information of the plan operator level and provides the training and prediction sets of label values such as startup time, total time, peak memory, and rows for machine learning models.

Table 13-33 GS_WLM_PLAN_ENCODING_TABLE columns

Name	Type	Description
queryid	bigint	Internal query ID used for statement execution
plan_node_id	integer	Plan node ID of the execution plan
parent_node_id	integer	Parent node ID of the operator
startup_time	bignit	Time when the operator starts to process the first data record
total_time	bigint	Total execution time of the operator, in ms
rows	bigint	Number of rows executed by the operator
peak_memory	integer	Maximum peak memory used by the operator on database nodes (unit: MB)
encode	text	Encoding information of the operator

13.2.34 GS_WLM_PLAN_OPERATOR_INFO

GS_WLM_PLAN_OPERATOR_INFO displays records about plan operator levels of completed jobs. The data is dumped from the kernel to the system catalog.

Table 13-34 GS_WLM_PLAN_OPERATOR_INFO columns

Name	Type	Description
datname	name	Name of the database where the collected plan information is located
queryid	bigint	Internal query ID used for statement execution
plan_node_id	integer	Plan node ID of the execution plan
startup_time	bigint	Time when the operator starts to process the first data record
total_time	bigint	Total execution time of the operator, in ms
actual_rows	bigint	Number of rows that are actually executed
max_peak_memory	integer	Maximum peak memory used by the operator on database nodes (unit: MB)
query_dop	integer	DOP of the operator
parent_node_id	integer	Parent node ID of the operator
left_child_id	integer	Left child node ID of the operator
right_child_id	integer	Right child node ID of the operator
operation	text	Name of the operation performed by the operator
orientation	text	Alignment mode of the operator
strategy	text	Implementation method of the operator
options	text	Selection mode of the operator
condition	text	Filter criteria of the operator
projection	text	Mapping relationship of the operator

13.2.35 GS_WLM_SESSION_QUERY_INFO_ALL

GS_WLM_SESSION_QUERY_INFO_ALL displays load management information about completed jobs executed on the current database instance. The current feature is a lab feature. Contact Huawei technical support before using it. The data is dumped from the kernel to the system catalog. If the GUC parameter **enable_resource_record** is set to **on**, query information in the kernel is imported to the system catalog **GS_WLM_SESSION_QUERY_INFO_ALL** every 3 minutes.

This system catalog can be queried by users with the **sysadmin** permission only in Postgres.

NOTE

If no data is displayed in the queried view, contact Huawei technical support.

Table 13-35 GS_WLM_SESSION_QUERY_INFO_ALL columns

Name	Type	Description
datid	oid	OID of the database the backend is connected to
dbname	text	Name of the database the backend is connected to
schemaname	text	Schema name
nodename	text	Name of the database instance where the statement is executed
username	text	Username used for connecting to the backend
application_name	text	Name of the application connected to the backend
client_addr	inet	IP address of the client connected to the backend. If this column is null, it indicates either that the client is connected via a Unix socket on the server machine or that this is an internal process such as autovacuum.
client_hostname	text	Host name of the connected client, as reported by a reverse DNS lookup of **client_addr**. This column will be non-null only for IP connections and only when **log_hostname** is enabled.
client_port	integer	TCP port number that the client uses for communication with the backend (**–1** if a Unix socket is used).
query_band	text	Job type, which is specified by the GUC parameter **query_band**. The default value is a null string.
block_time	bigint	Duration that the statement is blocked before being executed, including the statement parsing and optimization duration (unit: ms)
start_time	timestamp with time zone	Time when the statement execution starts
finish_time	timestamp with time zone	Time when the statement execution ends
duration	bigint	Execution time of a statement. The unit is ms.

Name	Type	Description
estimate_total_time	bigint	Estimated execution time of a statement. The unit is ms.
status	text	Final statement execution status, which can be **finished** (normal) or **aborted** (abnormal)
abort_info	text	Exception information displayed if the final statement execution status is **aborted**
resource_pool	text	Resource pool used by the user
control_group	text	Cgroup used by the statement
estimate_memory	integer	Estimated memory size of the statement
min_peak_memory	integer	Minimum memory peak of the statement across the database instances, in MB
max_peak_memory	integer	Maximum memory peak of the statement across the database instances, in MB
average_peak_memory	integer	Average memory usage during statement execution, in MB
memory_skew_percent	integer	Memory usage skew of the statement among the database instances
spill_info	text	Information about statement spill to the database instances: ● **None**: The statement has not been spilled to disks on the database instances. ● **All**: The statement has been spilled to disks on the database instances. ● *[a:b]*: The statement has been spilled to disks on *a* of *b* database instances.
min_spill_size	integer	Minimum spilled data among database instances when a spill occurs (unit: MB). The default value is **0**.
max_spill_size	integer	Maximum spilled data among database instances when a spill occurs (unit: MB). The default value is **0**.
average_spill_size	integer	Average spilled data among database instances when a spill occurs (unit: MB). The default value is **0**.
spill_skew_percent	integer	Database instance spill skew when a spill occurs
min_dn_time	bigint	Minimum execution time of the statement across the database instances (unit: ms)

Name	Type	Description
max_dn_time	bigint	Maximum execution time of the statement across the database instances (unit: ms)
average_dn_time	bigint	Average execution time of the statement across the database instances (unit: ms)
dntime_skew_percent	integer	Execution time skew of the statement among the database instances
min_cpu_time	bigint	Minimum CPU time of the statement across the database instances (unit: ms)
max_cpu_time	bigint	Maximum CPU time of the statement across the database instances (unit: ms)
total_cpu_time	bigint	Total CPU time of the statement across the database instances (unit: ms)
cpu_skew_percent	integer	CPU time skew of the statement among the database instances
min_peak_iops	integer	Minimum IOPS peak of the statement across the database instances. It is counted by ones in a column-store table and by ten thousands in a row-store table.
max_peak_iops	integer	Maximum IOPS peak of the statement across the database instances. It is counted by ones in a column-store table and by ten thousands in a row-store table.
average_peak_iops	integer	Average IOPS peak of the statement across the database instances. It is counted by ones in a column-store table and by ten thousands in a row-store table.
iops_skew_percent	integer	I/O skew of the statement among the database instances
warning	text	Warning. The following warnings are displayed: ● Spill file size large than 256MB ● Broadcast size large than 100MB ● Early spill ● Spill times is greater than 3 ● Spill on memory adaptive ● Hash table conflict
queryid	bigint	Internal query ID used for statement execution
query	text	Statement executed
query_plan	text	Execution plan of a statement

Name	Type	Description
node_group	text	Logical database instance of the user to which the statement belongs
cpu_top1_node_name	text	Name of the node with the highest CPU usage
cpu_top2_node_name	text	Name of the node with the second highest CPU usage
cpu_top3_node_name	text	Name of the node with the third highest CPU usage
cpu_top4_node_name	text	Name of the node with the fourth highest CPU usage
cpu_top5_node_name	text	Name of the node with the fifth highest CPU usage
mem_top1_node_name	text	Name of the node with the highest memory usage
mem_top2_node_name	text	Name of the node with the second highest memory usage
mem_top3_node_name	text	Name of the node with the third highest memory usage
mem_top4_node_name	text	Name of the node with the fourth highest memory usage
mem_top5_node_name	text	Name of the node with the fifth highest memory usage
cpu_top1_value	bigint	CPU usage
cpu_top2_value	bigint	CPU usage
cpu_top3_value	bigint	CPU usage
cpu_top4_value	bigint	CPU usage
cpu_top5_value	bigint	CPU usage
mem_top1_value	bigint	Memory usage
mem_top2_value	bigint	Memory usage
mem_top3_value	bigint	Memory usage

Name	Type	Description
mem_top4_value	bigint	Memory usage
mem_top5_value	bigint	Memory usage
top_mem_dn	text	Top *N* memory usage
top_cpu_dn	text	Top *N* CPU usage
n_returned_rows	bigint	Number of rows in the result set returned by the SELECT statement
n_tuples_fetched	bigint	Number of rows randomly scanned
n_tuples_returned	bigint	Number of rows sequentially scanned
n_tuples_inserted	bigint	Number of rows inserted
n_tuples_updated	bigint	Number of rows updated
n_tuples_deleted	bigint	Number of rows deleted
n_blocks_fetched	bigint	Number of cache loading times
n_blocks_hit	bigint	Cache hits
db_time	bigint	Valid DB time, which is accumulated if multiple threads are involved (unit: μs)
cpu_time	bigint	CPU time (unit: μs)
execution_time	bigint	Execution time in the executor (unit: μs)
parse_time	bigint	SQL parsing time (unit: μs)
plan_time	bigint	SQL plan generation time (unit: μs)
rewrite_time	bigint	SQL rewriting time (unit: μs)
pl_execution_time	bigint	Execution time of PL/pgSQL (unit: μs)
pl_compilation_time	bigint	Compilation time of PL/pgSQL (unit: μs)
net_send_time	bigint	Network time (unit: μs)
data_io_time	bigint	I/O time (unit: μs)

Name	Type	Description
is_slow_query	bigint	Whether the query is a slow query The value **1** indicates a slow query.

13.2.36 GS_WLM_USER_RESOURCE_HISTORY

GS_WLM_USER_RESOURCE_HISTORY stores information about resources used by users. Each record in this system catalog indicates resource usage of a user at a time point, including the memory, number of CPU cores, storage space, temporary space, operator flushing space, logical I/O traffic, number of logical I/O operations, and logical I/O rate. The memory, CPU, and I/O monitoring items record only resource usage of complex jobs. For I/O monitoring items, this parameter is valid only when **enable_logical_io_statistics** is set to **on**. The function of saving user monitoring data is enabled only when **enable_user_metric_persistent** is set to **on**. Data in the system catalog **GS_WLM_USER_RESOURCE_HISTORY** comes from the **PG_TOTAL_USER_RESOURCE_INFO** view. This system catalog can be queried by users with the **sysadmin** permission only in Postgres.

Table 13-36 GS_WLM_USER_RESOURCE_HISTORY

Name	Type	Description
username	text	Username
timestamp	timestamp with time zone	Timestamp
used_memory	integer	Size of the memory being used, in MB
total_memory	integer	Available memory, in MB The value **0** indicates that the available memory is not limited and depends on the maximum memory available in the database.
used_cpu	real	Number of CPU cores in use
total_cpu	integer	Total number of CPU cores of the Cgroup associated with the user on the node
used_space	bigint	Used storage space, in KB
total_space	bigint	Available storage space (unit: KB). The value **–1** indicates that the space is not limited.
used_temp_space	bigint	Used temporary storage space, in KB

Name	Type	Description
total_temp_space	bigint	Available temporary storage space, in KB. The value **–1** indicates that the maximum temporary storage space is not limited.
used_spill_space	bigint	Used space of operator flushing, in KB
total_spill_space	bigint	Available storage space for operator flushing, in KB. The value **–1** indicates that the maximum operator flushing space is not limited.
read_kbytes	bigint	Byte traffic of read operations in a monitoring period, in KB
write_kbytes	bigint	Byte traffic of write operations in a monitoring period, in KB
read_counts	bigint	Number of read operations in a monitoring period
write_counts	bigint	Number of write operations in a monitoring period
read_speed	real	Byte rate of read operations in a monitoring period, in KB/s
write_speed	real	Byte rate of write operations in a monitoring period, in KB/s

13.2.37 PG_AGGREGATE

PG_AGGREGATE records information about aggregate functions. Each entry in **PG_AGGREGATE** is an extension of an entry in **PG_PROC**. The **PG_PROC** entry carries the aggregate's name, input and output data types, and other information that is similar to ordinary functions.

Table 13-37 PG_AGGREGATE columns

Name	Type	Reference	Description
aggfnoid	regproc	**PG_PROC**.proname	**PG_PROC** proname of the aggregate function
aggtransfn	regproc	**PG_PROC**.proname	Transition function
aggcollectfn	regproc	**PG_PROC**.proname	Collect function
aggfinalfn	regproc	**PG_PROC**.proname	Final function (**0** if none)
aggsortop	oid	**PG_OPERATOR**.oid	Associated sort operator (**0** if none)

Name	Type	Reference	Description
aggtranstype	oid	PG_TYPE.oid	Data type of the aggregate function's internal transition (state) data The possible values and their meanings are defined by the types in **pg_type.h**. The main two types are polymorphic (isPolymorphicType) and non-polymorphic.
agginitval	text	-	Initial value of the transition state. This is a text column containing the initial value in its external string representation. If this column is null, the transition state value starts from null.
agginitcollect	text	-	Initial value of the collection state. This is a text column containing the initial value in its external string representation. If this column is null, the collection state value starts from null.
aggkind	"char"	-	Type of the aggregate function: ● **n**: normal aggregate ● **o**: ordered set aggregate
aggnumdirectargs	smallint	-	Number of direct parameters (non-aggregation-related parameters) of the aggregate function of the ordered set aggregate type. For an aggregate function of the normal aggregate type, the value is **0**.

13.2.38 PG_AM

PG_AM records information about index access methods. There is one row for each index access method supported by the system.

Table 13-38 PG_AM columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)
amname	name	-	Name of the access method
amstrategies	smallint	-	Number of operator strategies for the access method (**0** if the access method does not have a fixed set of operator strategies)
amsupport	smallint	-	Number of support routines for the access method
amcanorder	boolean	-	Whether the access method supports ordered scans sorted by the indexed column's value
amcanorderbyop	boolean	-	Whether the access method supports ordered scans sorted by the result of an operator on the indexed column
amcanbackward	boolean	-	Whether the access method supports backward scanning
amcanunique	boolean	-	Whether the access method supports unique indexes
amcanmulticol	boolean	-	Whether the access method supports multi-column indexes
amoptionalkey	boolean	-	Whether the access method supports scanning without any constraint for the first index column
amsearcharray	boolean	-	Whether the access method supports **ScalarArrayOpExpr** searches
amsearchnulls	boolean	-	Whether the access method supports **IS NULL/NOT NULL** searches
amstorage	boolean	-	Whether the index storage data type can differ from the column data type
amclusterable	boolean	-	Whether an index of this type can be clustered on
ampredlocks	boolean	-	Whether an index of this type manages fine-grained predicate locks

Name	Type	Reference	Description
amkeytype	oid	OID in **PG_TYPE**	Type of data stored in index (**0** if it is not a fixed type)
aminsert	regproc	**PG_PROC**.proname	"Insert this tuple" function
ambeginscan	regproc	**PG_PROC**.proname	"Prepare for index scan" function
amgettuple	regproc	**PG_PROC**.proname	"Next valid tuple" function (**0** if none)
amgetbitmap	regproc	**PG_PROC**.proname	"Fetch all valid tuples" function (**0** if none)
amrescan	regproc	**PG_PROC**.proname	"(Re)start index scan" function
amendscan	regproc	**PG_PROC**.proname	"Clean up after index scan" function
ammarkpos	regproc	**PG_PROC**.proname	"Mark current scan position" function
amrestrpos	regproc	**PG_PROC**.proname	"Restore marked scan position" function
ammerge	regproc	**PG_PROC**.proname	"Merge multiple indexes" function
ambuild	regproc	**PG_PROC**.proname	"Build new index" function
ambuildempty	regproc	**PG_PROC**.proname	"Build empty index" function
ambulkdelete	regproc	**PG_PROC**.proname	Bulk-delete function
amvacuumcleanup	regproc	**PG_PROC**.proname	Post-**VACUUM** cleanup function
amcanreturn	regproc	**PG_PROC**.proname	Function to check whether the index supports index-only scans (**0** if none)
amcostestimate	regproc	**PG_PROC**.proname	Function to estimate cost of an index scan
amoptions	regproc	**PG_PROC**.proname	Function to parse and validate **reloptions** for an index

13.2.39 PG_AMOP

PG_AMOP records information about operators associated with access method operator families. There is one row for each operator that is a member of an operator family. A family member can be either a search operator or an ordering operator. An operator can appear in more than one family, but cannot appear in more than one search position nor more than one ordering position within a family.

Table 13-39 PG_AMOP columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)
amopfamily	oid	**PG_OPFAMILY**.oid	Operator family of this entry
amoplefttype	oid	**PG_TYPE**.oid	Left-hand input data type of the operator For details about the possible values and their description, see **pg_type.h**.
amoprighttype	oid	**PG_TYPE**.oid	Right-hand input data type of the operator For details about the possible values and their description, see **pg_type.h**.
amopstrategy	smallint	-	Number of operator strategies
amoppurpose	"char"	-	Operator purpose, either **s** for search or **o** for ordering
amopopr	oid	**PG_OPERATOR**.oid	OID of the operator
amopmethod	oid	**PG_AM**.oid	Operator family of the index access method
amopsortfamily	oid	**PG_OPFAMILY**.oid	The B-tree operator family according to which this entry sorts for an ordering operator (**0** for a search operator)

A search operator entry indicates that an index of this operator family can be searched to find all rows satisfying **WHERE indexed_column operator constant**.

Obviously, such an operator must return a Boolean value, and its left-hand input type must match the index's column data type.

An ordering operator entry indicates that an index of this operator family can be scanned to return rows in the order represented by **ORDER BY indexed_column operator constant**. Such an operator could return any sortable data type, though again its left-hand input type must match the index's column data type. The exact semantics of **ORDER BY** are specified by the **amopsortfamily** column, which must reference the B-tree operator family for the operator's result type.

13.2.40 PG_AMPROC

PG_AMPROC records information about the support procedures associated with the access method operator families. There is one row for each support procedure that belongs to an operator family.

Table 13-40 PG_AMPROC columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)
amprocfamily	oid	**PG_OPFAMILY**.oid	Operator family of this entry
amproclefttype	oid	**PG_TYPE**.oid	Left-hand input data type of the associated operator For details about the possible values and their description, see **pg_type.h**.
amprocrighttype	oid	**PG_TYPE**.oid	Right-hand input data type of the associated operator For details about the possible values and their description, see **pg_type.h**.
amprocnum	smallint	-	Support procedure number
amproc	regproc	**PG_PROC** .proname	OID of the procedure

The usual interpretation of the **amproclefttype** and **amprocrighttype** columns is that they identify the left and right input types of the operator(s) that a particular support procedure supports. For some access methods, these match the input data type(s) of the support procedure itself; for others not. There is a notion of "default" support procedures for an index, which are those with **amproclefttype** and **amprocrighttype** both equal to the index opclass's **opcintype**.

13.2.41 PG_APP_WORKLOADGROUP_MAPPING

PG_APP_WORKLOADGROUP_MAPPING provides load mapping group information in the database.

Table 13-41 PG_APP_WORKLOADGROUP_MAPPING columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified)
appname	name	Application name
workload_gpname	name	Mapped workload group name

13.2.42 PG_ATTRDEF

PG_ATTRDEF records default values of columns.

Table 13-42 PG_ATTRDEF columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified)
adrelid	oid	Table to which a column belongs
adnum	smallint	Number of columns
adbin	pg_node_tree	Internal representation of the default value of a column or of a generated expression
adsrc	text	Internal representation of a readable default value or of a generated expression
adgencol	"char"	Specifies whether a column is a generated column. The value **s** indicates that the column is a generated column, and the value **'\0** indicates that the column is a common column. The default value is **'\0**.

13.2.43 PG_ATTRIBUTE

PG_ATTRIBUTE records information about table columns.

Table 13-43 PG_ATTRIBUTE columns

Name	Type	Description
attrelid	oid	Table to which a column belongs

Name	Type	Description
attname	name	Column name
atttypid	oid	Column type
attstattarget	integer	Level of details of statistics collected for this column by **ANALYZE**. ● The value **0** indicates that no statistics should be collected. ● A negative value indicates that the system default statistic object is used. ● The exact meaning of positive values is data type-dependent. For scalar data types, **attstattarget** is both the target number of "most common values" to collect, and the target number of histogram bins to create.
attlen	smallint	Copy of **pg_type.typlen** in the column's type
attnum	smallint	Number of the column
attndims	integer	Number of dimensions if the column is an array (**0** in other cases)
attcacheoff	integer	This column is always set to **–1** on disks. When it is loaded into a row descriptor in the memory, it may be updated to cache the offset of the columns in the row.
atttypmod	integer	Type-specific data supplied at the table creation time (for example, the maximum length of a **varchar** column). This column is used as the third parameter when passing to type-specific input functions and length coercion functions. The value will generally be **–1** for types that do not need **atttypmod**.
attbyval	boolean	Copy of **pg_type.typbyval** of this column's type
attstorage	"char"	Copy of **pg_type.typstorage** of this column's type
attalign	"char"	Copy of **pg_type.typalign** of this column's type
attnotnull	boolean	A non-null constraint. It is possible to change this column to enable or disable the constraint.
atthasdef	boolean	This column has a default value, in which case there will be a corresponding entry in the **PG_ATTRDEF** table that actually defines the value.
attisdropped	boolean	Indicates that this column has been deleted and is no longer valid. A deleted column is still physically present in the table but is ignored by the analyzer, so it cannot be accessed through SQL.

Name	Type	Description
attislocal	boolean	Indicates that this column is locally defined in the relationship. Note that a column can be locally defined and inherited simultaneously.
attcmprmode	tinyint	Compressed modes for a specific column. The compressed mode includes: ● ATT_CMPR_NOCOMPRESS ● ATT_CMPR_DELTA ● ATT_CMPR_DICTIONARY ● ATT_CMPR_PREFIX ● ATT_CMPR_NUMSTR
attinhcount	integer	Number of direct ancestors that this column has. A column with an ancestor cannot be dropped nor renamed.
attcollation	oid	Defined collation of a column
attacl	aclitem[]	Permissions for column-level access
attoptions	text[]	Column attribute. Currently, the following attributes are supported: **n_distinct**: number of **distinct** values of a column (excluding subtables). **n_distinct_inherited**: number of **distinct** values of a column (including subtables).
attfdwoptions	text[]	Column attribute of a foreign table. Currently, **dist_fdw**, **file_fdw**, and **log_fdw** do not use foreign table column attributes.
attinitdefval	bytea	**attinitdefval** stores the default value expression. **ADD COLUMN** in the row-store table must use this column.
attkvtype	tinyint	Specifies the key value type for a column. Value: 0. **ATT_KV_UNDEFINED**: default value 1. **ATT_KV_TAG**: dimension 2. **ATT_KV_FIELD**: indicator 3. **ATT_KV_TIMETAG**: time column

13.2.44 PG_AUTHID

PG_AUTHID records information about database authentication identifiers (roles). The concept of users is contained in that of roles. A user is actually a role whose **rolcanlogin** has been set. Any role, whether its **rolcanlogin** is set or not, can use other roles as members.

For GaussDB, only one **PG_AUTHID** exists, which is not available for every database. This system catalog is accessible only to system administrators.

Table 13-44 PG_AUTHID columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified)
rolname	name	Name of a role
rolsuper	boolean	Whether the role is the initial system administrator with the highest permission ● **t** (true): yes ● **f** (false): no
rolinherit	boolean	Whether the role automatically inherits permissions of roles of which it is a member ● **t** (true): automatically inherited ● **f** (false): not automatically inherited
rolcreaterole	boolean	Whether the role can create more roles ● **t** (true): yes ● **f** (false): no
rolcreatedb	boolean	Whether the role can create databases ● **t** (true): yes ● **f** (false): no
rolcatupdate	boolean	Whether the role can directly update system catalogs Only the initial system administrator whose **usesysid** is set to **10** has this permission. It is unavailable for other users. ● **t** (true): yes ● **f** (false): no
rolcanlogin	boolean	Whether the role can log in (whether this role can be given as the initial session authorization identifier) ● **t** (true): yes ● **f** (false): no
rolreplication	boolean	Whether the role has the replication permission ● **t** (true): yes ● **f** (false): no

Name	Type	Description
rolauditadmin	boolean	Whether the role has the audit administrator permission ● **t** (true): yes ● **f** (false): no
rolsystemadmin	boolean	Whether the role has system administrator permissions ● **t** (true): yes ● **f** (false): no
rolconnlimit	integer	Maximum number of concurrent connections that the role can make (valid for roles that can log in) The value **–1** indicates there is no limit.
rolpassword	text	Password (possibly encrypted); **NULL** if no password
rolvalidbegin	timestamp with time zone	Account validity start time (**NULL** if no start time)
rolvaliduntil	timestamp with time zone	Password expiry time (**NULL** if no expiration)
rolrespool	name	Resource pool that a user can use
roluseft	boolean	Whether the role can perform operations on foreign tables ● **t** (true): yes ● **f** (false): no
rolparentid	oid	OID of a group user to which the user belongs
roltabspace	text	Maximum size of a user data table
rolkind	"char"	Special user types, including private users and common users
rolnodegroup	oid	Unsupported currently
roltempspace	text	Maximum size of a user's temporary table, in KB
rolspillspace	text	Maximum size of data that can be written to disks when a user executes a job, in KB
rolexcpdata	text	Query rules that can be set by users (reserved)

Name	Type	Description
rolmonitoradmin	boolean	Whether the role has monitor administrator permissions ● **t** (true): yes ● **f** (false): no
roloperatoradmin	boolean	Whether the role has the O&M administrator permission ● **t** (true): yes ● **f** (false): no
rolpolicyadmin	boolean	Whether the role has the security policy administrator permission ● **t** (true): yes ● **f** (false): no

13.2.45 PG_AUTH_HISTORY

PG_AUTH_HISTORY records the authentication history of a role. This system catalog is accessible only to system administrators.

Table 13-45 PG_AUTH_HISTORY columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified)
roloid	oid	ID of a role
passwordtime	timestamp with time zone	Time of password creation and change
rolpassword	text	Ciphertext of the role password. The encryption mode is determined by the GUC parameter **password_encryption_type**.

13.2.46 PG_AUTH_MEMBERS

PG_AUTH_MEMBERS records the membership between roles.

Table 13-46 PG_AUTH_MEMBERS columns

Name	Type	Description
roleid	oid	ID of a role that has a member

Name	Type	Description
member	oid	ID of a role that is a member of ROLEID
grantor	oid	ID of a role that grants this membership
admin_option	boolean	Whether a member can grant membership in ROLEID to others

13.2.47 PG_CAST

PG_CAST records the conversion relationship between data types.

Table 13-47 PG_CAST columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
castsource	oid	OID of the source data type
casttarget	oid	OID of the target data type
castfunc	oid	OID of the conversion function (**0** if no conversion function is required)
castcontext	"char"	Conversion mode between the source and target data types. ● **e**: Only explicit conversion can be performed (using the CAST or :: syntax). ● **'i'**: Implicit conversion can be performed. ● **a**: Both explicit and implicit conversion can be performed between data types.
castmethod	"char"	Conversion method. ● **f**: Conversion is performed using the specified function in the **castfunc** column. ● **b**: Binary forcible conversion rather than the specified function in the **castfunc** column is performed between data types.

13.2.48 PG_CLASS

PG_CLASS records database objects and their relationship.

Table 13-48 PG_CLASS columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
relname	name	Name of an object, such as a table, index, or view
relnamespace	oid	OID of the namespace that contains the relationship
reltype	oid	Data type that corresponds to the table's row type. The index is 0 because the index does not have **PG_TYPE** records.
reloftype	oid	OID of the composite type (**0** for other types)
relowner	oid	Owner of the relationship
relam	oid	Access method used, such as B-tree and hash, if this is an index
relfilenode	oid	Name of the on-disk file of this relationship (**0** if such file does not exist)
reltablespace	oid	Tablespace in which this relationship is stored. If the value is **0**, the default tablespace in this database is used. This column is meaningless if the relationship has no on-disk file.
relpages	double precision	Size of the on-disk representation of the table in pages (of size **BLCKSZ**). This is only an estimate used by the optimizer.
reltuples	double precision	Number of rows in the table. This is only an estimate used by the optimizer.
relallvisible	integer	Number of pages marked as all visible in the table. This column is used by the optimizer for optimizing SQL execution. It is updated by **VACUUM**, **ANALYZE**, and a few DDL statements such as **CREATE INDEX**.
reltoastrelid	oid	OID of the TOAST table associated with the table (**0** if no TOAST table exists). The TOAST table stores large columns "offline" in a secondary table.
reltoastidxid	oid	OID of the index for a TOAST table (**0** for a table other than a TOAST table)
reldeltarelid	oid	OID of a Delta table Delta tables are attached to column-store tables. They store long tail data generated during storage data import.

Name	Type	Description
reldeltaidx	oid	OID of the index for a Delta table
relcudescrelid	oid	OID of a CU description table. CU description tables (Desc tables) belong to column-store tables. They control whether storage data in the HDFS table directory is visible.
relcudescidx	oid	OID of the index for a CU description table
relhasindex	boolean	Its value is **true** if this column is a table and has (or recently had) at least one index. It is set by **CREATE INDEX** but is not immediately cleared by **DROP INDEX**. If the **VACUUM** process detects that a table has no index, it clears the **relhasindex** column and sets the value to **false**.
relisshared	boolean	Its value is **true** if the table is shared across all database nodes in the database. Only certain system catalogs (such as **PG_DATABASE**) are shared.
relpersistence	"char"	● **p**: permanent table ● **u**: non-log table ● **g**: temporary table
relkind	"char"	● **r**: ordinary table ● **i**: index ● **I**: global index of a partitioned table ● **s**: sequence ● **L**: long sequence ● **v**: view ● **c**: composite type ● **t**: TOAST table ● **f**: foreign table ● **m**: materialized view
relnatts	smallint	Number of user columns in the relationship (excluding system columns). **PG_ATTRIBUTE** has the same number of rows as the user columns.
relchecks	smallint	Number of check constraints in the table. For details, see the system catalog **PG_CONSTRAINT**.
relhasoids	boolean	Its value is **true** if an OID is generated for each row of the relationship.
relhaspkey	boolean	Its value is **true** if the table has (or once had) a primary key.

Name	Type	Description
relhasrules	boolean	Its value is **true** if the table has rules. For details, see the system catalog **PG_REWRITE**.
relhastriggers	boolean	The value is **true** if the table has (or once had) triggers. Triggers of the table and view are recorded in the system catalog **PG_TRIGGER**.
relhassubclass	boolean	Its value is **true** if the table has (or once had) any inheritance child table.
relcmprs	tinyint	Whether the compression feature is enabled for the table. Note that only batch insertion triggers compression, so ordinary CRUD does not trigger compression. ● **0**: Tables that do not support compression (primarily system catalogs, on which the compression attribute cannot be modified). ● **1**: The compression feature of the table data is NOCOMPRESS or has no specified keyword. ● **2**: The compression feature of the table data is COMPRESS.
relhasclusterkey	boolean	Whether the local cluster storage is used
relrowmovement	boolean	Whether row migration is allowed when the partitioned table is updated. ● **true**: Row migration is allowed. ● **false**: Row migration is not allowed.
parttype	"char"	Whether the table or index has the property of a partitioned table. ● **p**: The table or index has the property of a partitioned table. ● **n**: The table or index does not have the property of a partitioned table. ● **v**: The table is a value partitioned table in HDFS. ● **s**: The table is a level-2 partitioned table.
relfrozenxid	xid32	All transaction IDs before this one have been replaced with a permanent ("frozen") transaction ID in the table. This column is used to track whether the table needs to be vacuumed to prevent transaction ID wraparound (or to allow **PG_CLOG** to be shrunk). The value is **0** (**InvalidTransactionId**) if the relationship is not a table. To ensure forward compatibility, this column is reserved. The **relfrozenxid64** column is added to record the information.

Name	Type	Description

relacl	aclitem[]	Access permissions.
		The command output of the query is as follows:
		`rolename=xxxx/yyyy  -- Assigning permissions to a role`
		`=xxxx/yyyy -- Assigning the permission to public`
		xxxx indicates assigned permissions, and *yyyy* indicates roles with the assigned permissions. For details on permission descriptions, see **Table 13-49**.

reloptions	text[]	Table or index access method, using character strings in the format of "keyword=value"
relreplident	"char"	Identifier of a decoding column in logical decoding. ● **d**: default (primary key, if any) ● **n**: none ● **f**: all columns ● **i**: The indisreplident of the index is specified or the default index is used.
relfrozenxid64	xid	All transaction IDs before this one have been replaced with a permanent ("frozen") transaction ID in the table. This column is used to track whether the table needs to be vacuumed to prevent transaction ID wraparound (or to allow **PG_CLOG** to be shrunk). The value is **0** (**InvalidTransactionId**) if the relationship is not a table. For a global temporary table, this field is meaningless. You can view **relfrozenxid64** of the global temporary table of each session in the **pg_catalog.pg_gtt_relstats** view.
relbucket	oid	Bucket information in **pg_hashbucket**.
relbucketkey	int2vector	Column number of a hash partition.
relminmxid	xid	All multi-transaction IDs before this one have been replaced with a transaction ID in the table. This column is used to track whether the table needs to be vacuumed in order to prevent multi-transaction ID wraparound or to allow **pg_clog** to be shrunk. The value is **0** (**InvalidTransactionId**) if the relationship is not a table.

Table 13-49 Description of permissions

Parameter	Parameter Description
r	SELECT (read)

Parameter	Parameter Description
w	UPDATE (write)
a	INSERT (insert)
d	DELETE
D	TRUNCATE
x	REFERENCES
t	TRIGGER
X	EXECUTE
U	USAGE
C	CREATE
c	CONNECT
T	TEMPORARY
A	ALTER
P	DROP
m	COMMENT
i	INDEX
v	VACUUM
*	Authorization options for preceding permissions

13.2.49 PG_COLLATION

PG_COLLATION describes available collations, which are essentially mappings from an SQL name to operating system locale categories.

Table 13-50 PG_COLLATION columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)
collname	name	-	Collation name (unique per namespace and encoding)
collnamespace	oid	OID in **PG_NAMESPACE**	OID of the namespace that contains this collation
collowner	oid	OID in **PG_AUTHID**	Owner of the collation

Name	Type	Reference	Description
collencoding	integer	-	Encoding in which the collation is applicable, or **–1** if it works for any encoding. It is compatible with PostgreSQL.
collcollate	name	-	**LC_COLLATE** for this collation object
collctype	name	-	**LC_CTYPE** for this collation object

13.2.50 PG_CONSTRAINT

PG_CONSTRAINT records check, primary key, and unique constraints on tables.

Table 13-51 PG_CONSTRAINT columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified)
conname	name	Constraint name (not necessarily unique)
connamespace	oid	OID of the namespace that contains the constraint
contype	"char"	● **c**: check constraint ● **p**: primary key constraint ● **u**: unique constraint ● **t**: trigger constraint ● **x**: mutual exclusion constraint ● **f**: foreign key constraint ● **s**: clustering constraint ● **i**: invalid constraint
condeferrable	boolean	Whether the constraint can be deferrable
condeferred	boolean	Whether the constraint can be deferrable by default
convalidated	boolean	Whether the constraint is valid Currently, it can be set to **false** only for foreign key and check constraints.
conrelid	oid	Table containing this constraint (**0** if it is not a table constraint)

Name	Type	Description
contypid	oid	Domain containing this constraint (**0** if it is not a domain constraint)
conindid	oid	ID of the index associated with the constraint
confrelid	oid	Referenced table if this constraint is a foreign key. Otherwise, the value is **0**.
confupdtype	"char"	Foreign key update action code ● **a**: no action ● **r**: restriction ● **c**: cascading ● **n**: The parameter is set to **null**. ● **d**: The default value is used.
confdeltype	"char"	Foreign key deletion action code ● **a**: no action ● **r**: restriction ● **c**: cascading ● **n**: The parameter is set to **null**. ● **d**: The default value is used.
confmatchtype	"char"	Foreign key match type ● **f**: full match ● **p**: partial match ● **u**: unspecified (The NULL value can be matched if **f** is specified.)
conislocal	boolean	Whether the local constraint is defined for the relationship
coninhcount	integer	Number of direct inheritance parent tables that this constraint has. When the value is not **0**, the constraint cannot be deleted or renamed.
connoinherit	boolean	Whether the constraint can be inherited
consoft	boolean	Whether the column indicates an informational constraint
conopt	boolean	Whether you can use the informational constraint to optimize the execution plan
conkey	smallint[]	Column list of the constrained control if this column is a table constraint
confkey	smallint[]	List of referenced columns if this column is a foreign key.

Name	Type	Description
conpfeqop	oid[]	ID list of the equality operators for PK = FK comparisons if this column is a foreign key.
conppeqop	oid[]	ID list of the equality operators for PK = PK comparisons if this column is a foreign key.
conffeqop	oid[]	ID list of the equality operators for FK = FK comparisons if this column is a foreign key. The value is empty because foreign keys are not supported currently.
conexclop	oid[]	ID list of the per-column exclusion operators if this column is an exclusion constraint
conbin	pg_node_tree	Internal representation of the expression if this column is a check constraint
consrc	text	Human-readable representation of the expression if this column is a check constraint
conincluding	smallint[]	Not for constraint, but will be included in the attribute column of **INDEX**.

NOTICE

- **consrc** is not updated when referenced objects change and does not track new column names. Instead of relying on this column to update, you are advised to use **pg_get_constraintdef()** to extract the definition of a check constraint.
- **pg_class.relchecks** must agree with the number of check-constraint entries found in the table for each relationship.

13.2.51 PG_CONVERSION

PG_CONVERSION describes encoding conversion information.

Table 13-52 PG_CONVERSION columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)
conname	name	-	Conversion name (unique within a namespace)
connamespace	oid	PG_NAMESPACE.oid	OID of the namespace that contains this conversion

Name	Type	Reference	Description
conowner	oid	PG_AUTHID.oid	Owner of the conversion
conforencoding	integer	-	Source encoding ID
contoencoding	integer	-	Destination encoding ID
conproc	regproc	PG_PROC.proname	Conversion procedure
condefault	boolean	-	Whether the default conversion is used

13.2.52 PG_DATABASE

PG_DATABASE records information about available databases.

Table 13-53 PG_DATABASE columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
datname	name	Database name
datdba	oid	Owner of the database, usually the user who created it
encoding	integer	Character encoding for the database
datcollate	name	Sequence used by the database
datctype	name	Character type used by the database
datistemplate	boolean	Whether the database can be used as a template database
datallowconn	boolean	If the value is **false**, no one can connect to this database. This column is used to protect the **template0** database from being altered.
datconnlimit	integer	Maximum number of concurrent connections allowed on this database. The value **–1** indicates no limit.
datlastsysoid	oid	Last system OID in the database

Name	Type	Description
datfrozenxid	xid32	Tracks whether the database needs to be vacuumed to prevent transaction ID wraparound. This column is discarded in the current version. To ensure forward compatibility, this column is reserved. The **datfrozenxid64** column is added to record the information.
dattablespace	oid	Default tablespace of the database
datcompatibility	name	Database compatibility mode. Currently, four compatibility modes are supported: A, B, C, and PG, indicating that the Oracle, MySQL, Teradata, and Postgres databases are compatible.
datacl	aclitem[]	Access permissions
datfrozenxid64	xid	Tracks whether the database needs to be vacuumed to prevent transaction ID wraparound.
datminmxid	xid	All multi-transaction IDs before this one have been replaced with a transaction ID in the database. This is used to track whether the database needs to be vacuumed in order to prevent transaction IDs wraparound (or to allow **pg_clog** to be shrunk). It is the minimum **pg_class.relminmxid** value of all tables in the database.

13.2.53 PG_DB_ROLE_SETTING

PG_DB_ROLE_SETTING records the default values of configuration items bound to each role and database when the database is running.

Table 13-54 PG_DB_ROLE_SETTING columns

Name	Type	Description
setdatabase	oid	Database corresponding to the configuration items (**0** if no database is specified)
setrole	oid	Role corresponding to the configuration items (**0** if no role is specified)
setconfig	text[]	Default value of the configuration item during running. For details about the configuration method, see **Table 10-2**.

13.2.54 PG_DEFAULT_ACL

PG_DEFAULT_ACL records initial permissions assigned to newly created objects.

Table 13-55 PG_DEFAULT_ACL columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified)
defaclrole	oid	ID of the role associated with the permission
defaclnamespace	oid	Namespace associated with the permission (**0** if no ID)
defaclobjtype	"char"	Object type of the permission ● **r** indicates a table or view. ● **S** indicates a sequence. ● **f** indicates a function. ● **T** indicates a type. ● **K** indicates the client master key. ● **k** indicates the column encryption key.
defaclacl	aclitem[]	Access permissions that this type of object should have on creation

13.2.55 PG_DEPEND

PG_DEPEND records the dependency between database objects. This information allows **DROP** commands to find which other objects must be dropped by **DROP CASCADE** or prevent dropping in the **DROP RESTRICT** case.

See also **PG_SHDEPEND**, which performs a similar function for dependencies involving objects that are shared across databases.

Table 13-56 PG_DEPEND columns

Name	Type	Reference	Description
classid	oid	OID in **PG_CLASS**	OID of the system catalog where a dependent object resides
objid	oid	Any OID column	OID of the dependent object
objsubid	integer	-	For a table column, this is the column number (**objid** and **classid** refer to the table itself). The value is **0** for all other object types.

Name	Type	Reference	Description
refclassid	oid	OID in PG_CLASS	OID of the system catalog where a referenced object resides
refobjid	oid	Any OID column	OID of the referenced object
refobjsubid	integer	-	For a table column, this is the column number (**refobjid** and **refclassid** refer to the table itself). The value is **0** for all other object types.
deptype	"char"	-	A code defining the specific semantics of this dependency

In all cases, a **PG_DEPEND** entry indicates that the referenced object cannot be dropped without dropping the dependent object. However, there are several subflavors identified by **deptype**:

- DEPENDENCY_NORMAL (n): A normal relationship between separately created objects. The dependent object can be dropped without affecting the referenced object. The referenced object can only be dropped by specifying **CASCADE**, in which case the dependent object is dropped too. Example: a table column has a normal dependency on its data type.
- DEPENDENCY_AUTO (a): The dependent object can be dropped separately from the referenced object, and should be automatically dropped (regardless of RESTRICT or CASCADE mode) if the referenced object is dropped. Example: a named constraint on a table is made autodependent on the table, so that it will go away if the table is dropped.
- DEPENDENCY_INTERNAL (i): The dependent object was created as part of creation of the referenced object, and is only a part of its internal implementation. A DROP of the dependent object will be disallowed outright (We'll tell the user to issue a DROP against the referenced object, instead). A **DROP** of the referenced object will be propagated through to drop the dependent object whether **CASCADE** is specified or not.
- DEPENDENCY_EXTENSION (e): The dependent object is a member of the extension of the referenced object (see **PG_EXTENSION**). The dependent object can be dropped only via **DROP EXTENSION** on the referenced object. Functionally this dependency type acts the same as an internal dependency, but it is kept separate for clarity and to simplify **GS_DUMP**.
- DEPENDENCY_PIN (p): There is no dependent object; this type of entry is a signal that the system itself depends on the referenced object, and so that object must never be deleted. Entries of this type are created only by **initdb**. The columns for the dependent object contain zeroes.

13.2.56 PG_DESCRIPTION

PG_DESCRIPTION records optional descriptions (comments) for each database object. Descriptions of many built-in system objects are provided in the initial contents of **PG_DESCRIPTION**.

See also **PG_SHDESCRIPTION**, which provides a similar function for descriptions involving objects that are shared within the entire database.

Table 13-57 PG_DESCRIPTION columns

Name	Type	Reference	Description
objoid	oid	Any OID column	OID of the object that this description pertains to
classoid	oid	**PG_CLASS**.oid	OID of the system catalog where the object appears
objsubid	integer	-	Column number for a comment on a table column (**objoid** and **classoid** refer to the table itself); **0** for all other object types
description	text	-	Arbitrary text that serves as the description of the object

13.2.57 PG_DIRECTORY

PG_DIRECTORY stores directories added by users. You can run the **CREATE DIRECTORY** statement to add directories to the system catalog. Currently, only system administrators can perform this operation.

Table 13-58 PG_DIRECTORY columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified)
dirname	name	Name of a directory object
owner	oid	Owner of a directory object
dirpath	text	Directory path.
diracl	aclitem[]	Access permissions.

13.2.58 PG_ENUM

PG_ENUM contains entries showing the values and labels for each enumerated type. The internal representation of a given enumerated value is actually the OID of its associated row in **PG_ENUM**.

Table 13-59 PG_ENUM columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)
enumtypid	oid	**PG_TYPE**.oid	OID of the **PG_TYPE** entry owning this enumerated value
enumsortorder	real	-	Sort position of this enumerated value within its enumerated type
enumlabel	name	-	Textual label for this enumerated value

The OIDs for **PG_ENUM** rows follow a special rule: even-numbered OIDs are guaranteed to be ordered in the same way as the sort ordering of their enumerated type. If two even OIDs belong to the same enumerated type, the smaller OID must have the smaller **enumsortorder** value. Odd-numbered OID values need bear no relationship to the sort order. This rule allows the enumerated comparison routines to avoid catalog lookups in many common cases. The routines that create and alter enumerated types attempt to assign even OIDs to enumerated values whenever possible.

When an enumerated type is created, its members are assigned sort-order positions from 1 to *n*. However, members added later might be given negative or fractional values of **enumsortorder**. The only requirement on these values is that they be correctly ordered and unique within each enumerated type.

13.2.59 PG_EXTENSION

PG_EXTENSION records information about the installed extensions. Default extensions of GaussDB: PLPGSQL.

Table 13-60 PG_EXTENSION

Name	Type	Description
oid	oid	Database object ID
extname	name	Extension name
extowner	oid	Owner of the extension
extnamespace	oid	Namespace containing the extension's exported objects
extrelocatable	boolean	Whether the extension can be relocated to another namespace. Value **true** indicates that relocation is allowed.
extversion	text	Version number of the extension

Name	Type	Description
extconfig	oid[]	Configuration information about the extension
extcondition	text[]	Filter conditions for the extension's configuration information

13.2.60 PG_EXTENSION_DATA_SOURCE

PG_EXTENSION_DATA_SOURCE records information about external data sources. An external data source contains information about an external database, such as its password encoding. It is mainly used with Extension Connector. The current feature is a lab feature. Contact Huawei technical support before using it.

Table 13-61 PG_EXTENSION_DATA_SOURCE columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)
srcname	name	-	Name of an external data source
srcowner	oid	PG_AUTHID.oid	Owner of a foreign data source
srctype	text	-	Type of an external data source. It is **NULL** by default.
srcversion	text	-	Version of an external data source. It is **NULL** by default.
srcacl	aclitem[]	-	Access permissions.
srcoptions	text[]	-	Option used for foreign data sources, expressed in a string in the format of keyword=value

13.2.61 PG_FOREIGN_DATA_WRAPPER

PG_FOREIGN_DATA_WRAPPER records foreign-data wrapper definitions. A foreign-data wrapper is the mechanism by which external data, residing on foreign servers, is accessed.

Table 13-62 PG_FOREIGN_DATA_WRAPPER columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)
fdwname	name	-	Name of a foreign-data wrapper
fdwowner	oid	OID in **PG_AUTHID**	Owner of a foreign-data wrapper
fdwhandler	oid	OID in **PG_PROC**	References a handler function that is responsible for supplying execution routines for a foreign-data wrapper The value is **0** if no handler is provided.
fdwvalidator	oid	OID in **PG_PROC**	References a validator function that is responsible for checking the validity of the options given to a foreign-data wrapper, as well as options for foreign servers and user mappings using the foreign-data wrapper. The value is **0** if no validator is provided.
fdwacl	aclitem[]	-	Access permissions
fdwoptions	text[]	-	Foreign-data wrapper specific option, expressed in a string in the format of keyword=value

13.2.62 PG_FOREIGN_SERVER

PG_FOREIGN_SERVER records foreign server definitions. A foreign server describes a source of external data, such as a remote server. Foreign servers are accessed via foreign-data wrappers.

Table 13-63 PG_FOREIGN_SERVER columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)
srvname	name	-	Name of a foreign server
srvowner	oid	OID in **PG_AUTHID**	Owner of the foreign server
srvfdw	oid	OID in **PG_FOREIGN_DATA_WRAPPER**	OID of the foreign-data wrapper on a foreign server

Name	Type	Reference	Description
srvtype	text	-	Type of the server (optional)
srvversion	text	-	Version of the server (optional)
srvacl	aclitem[]	-	Access permissions
srvoptions	text[]	-	Option used for foreign servers, expressed in a string in the format of keyword=value

13.2.63 PG_FOREIGN_TABLE

PG_FOREIGN_TABLE records auxiliary information about foreign tables.

Table 13-64 PG_FOREIGN_TABLE columns

Name	Type	Description
ftrelid	oid	ID of a foreign table
ftserver	oid	Server where a foreign table is located
ftwriteonly	boolean	Specifies whether data can be written in a foreign table. Value range: ● **t** (true): yes ● **f** (false): no
ftoptions	text[]	Options of a foreign table. For details, see the description of **CREATE FOREIGN TABLE**.

13.2.64 PG_HASHBUCKET

PG_HASHBUCKET records hash bucket information.

Table 13-65 PG_HASHBUCKET columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified)
bucketid	oid	Hash value calculated for a bucket vector. The hash value can be used to accelerate the search for a bucket vector.
bucketcnt	integer	Number of shards

Name	Type	Description
bucketmapsize	integer	Total number of shards on all DNs
bucketref	integer	Reserved column with **1** as its default value
bucketvector	oidvector_extend	Records all bucket IDs contained in the bucket information in this row. A unique index is created in this column. Tables with the same bucket ID share the **PG_HASHBUCKET** data in the same row.

13.2.65 PG_INDEX

PG_INDEX records part of index information. The rest is mostly recorded in **PG_CLASS**.

Table 13-66 PG_INDEX columns

Name	Type	Description
indexrelid	oid	OID of the **PG_CLASS** entry for the index
indrelid	oid	OID of the **PG_CLASS** entry for the table that uses the index
indnatts	smallint	Number of columns in the index
indisunique	boolean	This index is a unique index if the value is **true**. This index is not a unique index if the value is **false**.
indisprimary	boolean	Primary key of the table if the value is **true**. **indisunique** should always be **true** when the value of this column is **true**. This index is not the primary key of the table if the value is **false**.
indisexclusion	boolean	Whether the index supports exclusion constraints This index does not support exclusion constraints if the value is **false**.
indimmediate	boolean	Whether a uniqueness check is performed upon data insertion A uniqueness check is not performed upon data insertion if the value is **false**.
indisclustered	boolean	Whether the table was last clustered on the index The table is not clustered on this index if the value is **false**.

Name	Type	Description
indisusable	boolean	This index supports INSERT and SELECT if the value is **true**. This index does not support INSERT and SELECT if the value is **false**.
indisvalid	boolean	Whether the index is valid for queries. If this column is **false**, the index is possibly incomplete and must still be modified by INSERT/UPDATE operations, but it cannot safely be used for queries. If it is a unique index, the uniqueness property is also not **true**.
indcheckxmin	boolean	If the value is **true**, queries must not use the index until the xmin of this row in **PG_INDEX** is below their **TransactionXmin**, because the table may contain broken HOT chains with incompatible rows that they can see. If the value is **false**, queries can use indexes.
indisready	boolean	The index is available for inserted data if the value is **true**. Otherwise, this index is ignored when data is inserted or modified.
indkey	int2vector	This is an array of **indnatts** values indicating that this index creates table columns. For example, a value of **1 3** indicates that the first and the third columns make up the index key. The value **0** in this array indicates that the corresponding index attribute is an expression over the table columns, rather than a simple column reference.
indcollation	oidvector	ID of each column used by the index
indclass	oidvector	For each column in the index key, this contains the OID of the operator class to use. See **PG_OPCLASS** for details.
indoption	int2vector	Array of values that store per-column flag bits. The meaning of the bits is defined by the index's access method.
indexprs	pg_node_tree	Expression trees (in **nodeToString()** representation) for index attributes that are not simple column references. It is a list with one element for each zero entry in **INDKEY**. The value is null if all index attributes are simple references.
indpred	pg_node_tree	Expression tree (in **nodeToString()** representation) for partial index predicate. If the index is not a partial index, this column is an empty string.

Name	Type	Description
indisreplident	boolean	If the value is **true**, the column of this index becomes the decoded column of logical decoding. If the value is **false**, the column of this index is not the decoded column of logical decoding.
indnkeyatts	smallint	Total number of columns in the index. The columns that exceed the value of **indnatts** are not involved in the index query.

13.2.66 PG_INHERITS

PG_INHERITS records information about table inheritance hierarchies. There is one entry for each direct child table in the database. Indirect inheritance can be determined by following chains of entries.

Table 13-67 PG_INHERITS columns

Name	Type	Reference	Description
inhrelid	oid	**PG_CLASS**.oid	OID of a child table
inhparent	oid	**PG_CLASS**.oid	OID of a parent table
inhseqno	integer	-	If there is more than one direct parent for a child table (multiple inheritances), this number tells the order in which the inherited columns are to be arranged. The count starts at 1.

13.2.67 PG_JOB

PG_JOB records detailed information about jobs created by users. Dedicated threads poll the system catalog **PG_JOB** and trigger jobs based on scheduled job execution time, and update job status in **PG_JOB**. This system catalog belongs to the Shared Relation category. All job records are visible to all databases.

Table 13-68 PG_JOB columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified)
job_id	bigint	Job ID, primary key, unique (with a unique index)

Name	Type	Description
current_postgres_pid	bigint	If the current job has been executed, the GaussDB thread ID of this job is recorded. The default value is **–1**, indicating that the job has not yet been executed.
log_user	name	Username of the job creator
priv_user	name	Username of the job executor
dbname	name	Name of the database in which the job will be executed
node_name	name	Primary database node on which the job will be executed
job_status	"char"	Status of the job. The value can be **r**, **s**, **f**, or **d**. The default value is **s**. The indications are as follows: Status of job step: r=running, s=successfully finished, f=job failed, d=disable If a job fails to be executed for 16 consecutive times, **job_status** is automatically set to **d**, and no more attempt will be made on this job. Note: When you disable a scheduled task (by setting **job_queue_processes** to **0**), the thread that monitors the job execution is not started, and the job status will not be updated. You can ignore this status. Only when the scheduled task function is enabled (**job_queue_processes** is not set to **0**), the system updates the value of this column based on the real-time job status.
start_date	timestamp without time zone	Start time of the first job execution, accurate to millisecond
next_run_date	timestamp without time zone	Scheduled time of the next job execution, accurate to millisecond
failure_count	smallint	Number of times the job has started and failed. If a job fails to be executed for 16 consecutive times, no more attempt will be made on it.
interval	text	Job execution interval
last_start_date	timestamp without time zone	Start time of the last job execution, accurate to millisecond
last_end_date	timestamp without time zone	End time of the last job execution, accurate to millisecond

Name	Type	Description
last_suc_date	timestamp without time zone	Start time of the last successful job execution, accurate to millisecond
this_run_date	timestamp without time zone	Start time of the ongoing job execution, accurate to millisecond
nspname	name	Name of the schema used for job execution
job_name	text	Name of the DBE_SCHEDULER scheduled task.
end_date	timestamp without time zone	Expiration time of the DBE_SCHEDULER scheduled task, accurate to millisecond.
enable	boolean	The DBE_SCHEDULER scheduled task enabling status. The options are as follows: ● **true**: enabled ● **false**: disabled
failure_msg	text	Error information about the latest task execution.

13.2.68 PG_JOB_PROC

PG_JOB_PROC records the content of each job in the **PG_JOB** table, including the PL/SQL code blocks and anonymous blocks. Storing such information in the system catalog **PG_JOB** and loading it to the shared memory will result in excessive memory usage. Therefore, such information is stored in a separate table and is retrieved when needed.

Table 13-69 PG_JOB_PROC columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified)
job_id	integer	Foreign key, which is associated with **job_id** in the system catalog **PG_JOB**
what	text	Job content, which is the program content in the DBE_SCHEDULER scheduled task.
job_name	text	Name of the DBE_SCHEDULER scheduled task or program.

13.2.69 PG_LANGUAGE

PG_LANGUAGE registers programming languages. You can use them and interfaces to write functions or stored procedures.

Table 13-70 PG_LANGUAGE columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)
lanname	name	-	Language name
lanowner	oid	**PG_AUTHID**.oid	Owner of the language
lanispl	boolean	-	The value is **false** for internal languages (such as SQL) and **true** for user-defined languages. Currently, **gs_dump** still uses this column to determine which languages need to be dumped, but this might be replaced by a different mechanism in the future.
lanpltrusted	boolean	-	The value is **true** if this is a trusted language, which means that it is believed not to grant access to anything outside the normal SQL execution environment. Only the initial user can create functions in untrusted languages.
lanplcallfoid	oid	**PG_PROC**.oid	For non-internal languages, this column references the language handler, which is a special function responsible for executing all functions that are written in the particular language.
laninline	oid	**PG_PROC**.oid	This column references a function responsible for executing "inline" anonymous code blocks (DO blocks). The value is **0** if inline blocks are not supported.
lanvalidator	oid	**PG_PROC**.oid	This column references a language validator function responsible for checking the syntax and validity of new functions when they are created. The value is **0** if no validator is provided.
lanacl	aclitem[]	-	Access permissions

13.2.70 PG_LARGEOBJECT

PG_LARGEOBJECT records data making up large objects. A large object is identified by an OID assigned when it is created. Each large object is broken into segments or "pages" small enough to be conveniently stored as rows in **PG_LARGEOBJECT**. The amount of data per page is defined as **LOBLKSIZE**.

This system catalog is accessible only to system administrators.

Table 13-71 PG_LARGEOBJECT columns

Name	Type	Reference	Description
loid	oid	**PG_LARGEOBJECT_METADATA**.oid	Identifier of the large object that includes this page
pageno	integer	-	Page number of this page within its large object (counting from zero)
data	bytea	-	Data stored in the large object. This will never be more than **LOBLKSIZE** bytes and might be less.

Each row of **PG_LARGEOBJECT** holds data for one page of a large object, beginning at byte offset (**pageno * LOBLKSIZE**) within the object. The implementation allows sparse storage: pages might be missing, and might be shorter than **LOBLKSIZE** bytes even if they are not the last page of the object. Missing regions within a large object read as zeroes.

13.2.71 PG_LARGEOBJECT_METADATA

PG_LARGEOBJECT_METADATA records metadata associated with large objects. The actual large object data is stored in **PG_LARGEOBJECT**.

Table 13-72 PG_LARGEOBJECT_METADATA columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)
lomowner	oid	**PG_AUTHID**.oid	Owner of the large object
lomacl	aclitem[]	-	Access permissions

13.2.72 PG_NAMESPACE

PG_NAMESPACE records namespaces, that is, schema-related information.

Table 13-73 PG_NAMESPACE columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified)
nspname	name	Name of a namespace
nspowner	oid	Owner of a namespace
nsptimeline	bigint	Timeline when a namespace is created on the database node. This column is for internal use and valid only on the database node.
nspacl	aclitem[]	Access permission
in_redistribution	"char"	Specifies whether the content is in the redistribution state.
nspblockchain	boolean	● If the value is **true**, the tamper-proof schema is used. ● If the value is **false**, the non-tamper-proof schema is used.

13.2.73 PG_OBJECT

PG_OBJECT records the creator, creation time, and last modification time of objects of specified types (ordinary tables, indexes, sequences, views, stored procedures, and functions).

Table 13-74 PG_OBJECT columns

Name	Type	Description
object_oid	oid	Object identifier
object_type	"char"	Object type ● **r**: ordinary table ● **i**: index ● **s**: sequence ● **v**: view ● **p**: stored procedure and function
creator	oid	ID of a creator

Name	Type	Description
ctime	timestamp with time zone	Creation time of an object
mtime	timestamp with time zone	Last modification time of an object. The modification operations include **ALTER**, **GRANT**, and **REVOKE**.
createcsn	bigint	CSN when an object is created
changecsn	bigint	CSN when DDL operations are performed on a table or an index

NOTICE

- Objects created or modified during database initialization (initdb) cannot be recorded. **PG_OBJECT** does not contain these object records.
- When an object created before the upgrade is modified again, the modification time (specified by **mtime**) is recorded. When DDL operations are performed on a table or an index, the transaction commit sequence number (specified by **changecsn**) of the transaction to which the table or index belongs is recorded. Because the creation time of the object cannot be obtained, **ctime** and **createcsn** are empty.
- The time recorded by **ctime** and **mtime** is the start time of the transaction to which the current operation belongs.
- The time of object modification due to capacity expansion is also recorded.
- **createcsn** and **changecsn** record the transaction commit sequence number of the transaction to which the current operation belongs.
- When **enable_gtt_concurrent_truncate** is set to **on**, the **mtime** field is not updated when the global temporary table is truncated.

13.2.74 PG_OBSSCANINFO

PG_OBSSCANINFO defines OBS runtime information scanned in database instance acceleration scenarios. Each record corresponds to a piece of runtime information of a foreign table on OBS in a query. (Due to specification changes, the current version no longer supports the current feature. Do not use this feature.)

Table 13-75 PG_OBSSCANINFO columns

Name	Type	Reference	Description
query_id	bigint	-	Query ID
user_id	text	-	Database user who performs the query

Name	Type	Reference	Description
table_name	text	-	Name of a foreign table on OBS
file_type	text	-	Format of the file that stores underlying data
time_stamp	timestamp with time zone	-	Scanning start time
actual_time	double precision	-	Scanning execution time, in seconds
file_scanned	bigint	-	Number of files scanned
data_size	double precision	-	Size of data scanned, in bytes
billing_info	text	-	Reserved column

13.2.75 PG_OPCLASS

PG_OPCLASS defines index access method operator classes.

Each operator class defines semantics for index columns of a particular data type and a particular index access method. An operator class essentially specifies that a particular operator family is applicable to a particular indexable column data type. The set of operators from the family that are actually usable with the indexed column are whichever ones accept the column's data type as their left-hand input.

Table 13-76 PG_OPCLASS columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)
opcmethod	oid	OID in **PG_AM**	Index access method operator class served by an operator class
opcname	name	-	Name of the operator class
opcnamespace	oid	OID in **PG_NAMESPACE**	Namespace of the operator class
opcowner	oid	OID in **PG_AUTHID**	Owner of the operator class
opcfamily	oid	OID in **PG_OPFAMILY**	Operator family containing the operator class
opcintype	oid	OID in **PG_TYPE**	Data type that the operator class indexes

Name	Type	Reference	Description
opcdefault	boolean	-	The value is **true** if this operator class is the default for **opcintype**.
opckeytype	oid	OID in **PG_TYPE**	Type of data stored in an index, or zero if same as **opcintype**

An operator class's **opcmethod** must match the **opfmethod** of its containing operator family.

13.2.76 PG_OPERATOR

PG_OPERATOR records information about operators.

Table 13-77 PG_OPERATOR columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)
oprname	name	-	Name of an operator
oprnamespace	oid	OID in **PG_NAMESPACE**	OID of the namespace that contains the operator
oprowner	oid	OID in **PG_AUTHID**	Owner of the operator
oprkind	"char"	-	• **b**: infix ("both") • **l**: prefix ("left") • **r**: postfix ("right")
oprcanmerge	boolean	-	Whether the operator supports merge joins • **t (true)**: yes • **f (false)**: no
oprcanhash	boolean	-	Whether the operator supports hash joins • **t (true)**: yes • **f (false)**: no
oprleft	oid	OID in **PG_TYPE**	Type of the left operand
oprright	oid	OID in **PG_TYPE**	Type of the right operand
oprresult	oid	OID in **PG_TYPE**	Type of the result

Name	Type	Reference	Description
oprcom	oid	OID in **PG_OPERATOR**	Commutator of this operator, if any
oprnegate	oid	OID in **PG_OPERATOR**	Negator of this operator, if any
oprcode	regproc	**PG_PROC**.proname	Function that implements the operator
oprrest	regproc	**PG_PROC**.proname	Restriction selectivity estimation function for the operator
oprjoin	regproc	**PG_PROC**.proname	Join selectivity estimation function for the operator

13.2.77 PG_OPFAMILY

PG_OPFAMILY defines operator families.

Each operator family is a collection of operators and associated support routines that implement semantics specified for a particular index access method. Furthermore, the operators in a family are all compatible, in a way that is specified by the access method. The operator family allows cross-data-type operators to be used with indexes and to be reasoned about using knowledge of access method semantics.

Table 13-78 PG_OPFAMILY columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)
opfmethod	oid	**PG_AM**.oid	Index access method used by an operator family
opfname	name	-	Name of the operator family
opfnamespace	oid	**PG_NAMESPACE**.oid	Namespace of the operator family
opfowner	oid	**PG_AUTHID**.oid	Owner of the operator family

The majority of the information defining an operator family is not in its **PG_OPFAMILY** row, but in the associated rows in **PG_AMOP**, **PG_AMPROC**, and **PG_OPCLASS**.

13.2.78 PG_PARTITION

PG_PARTITION records all partitioned tables, table partitions, TOAST tables on table partitions, and index partitions in the database. Partitioned index information is not stored in the system catalog **PG_PARTITION**.

Table 13-79 PG_PARTITION columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
relname	name	Names of the partitioned tables, table partitions, TOAST tables on table partitions, and index partitions
parttype	"char"	Object type ● **r**: partitioned table ● **p**: table partition ● **s**: table subpartition ● **x**: index partition ● **t**: TOAST table
parentid	oid	OID of the partitioned table in **PG_CLASS** when the object is a partitioned table or table partition OID of the partitioned index when the object is an index partition
rangenum	integer	Reserved
intervalnum	integer	Reserved
partstrategy	"char"	Partition policy of the partitioned table ● **r**: range partition ● **v**: numeric partition ● **i**: interval partition ● **l**: list partition ● **h**: hash partition ● **n**: invalid partition
relfilenode	oid	Physical storage locations of the table partition, index partition, and TOAST table on the table partition
reltablespace	oid	OID of the tablespace containing the table partition, index partition, and TOAST table on the table partition
relpages	double precision	Statistics: numbers of data pages of the table partition and index partition

Name	Type	Description
reltuples	double precision	Statistics: numbers of tuples of the table partition and index partition
relallvisible	integer	Statistics: number of visible data pages of the table partition and index partition
reltoastrelid	oid	OID of the TOAST table corresponding to the table partition
reltoastidxid	oid	OID of the TOAST table index corresponding to the table partition
indextblid	oid	OID of the table partition corresponding to the index partition
indisusable	boolean	Whether the index partition is available
reldeltarelid	oid	OID of a Delta table
reldeltaidx	oid	OID of the index for a Delta table
relcudescrelid	oid	OID of a CU description table
relcudescidx	oid	OID of the index for a CU description table
relfrozenxid	xid32	Frozen transaction ID To ensure forward compatibility, this column is reserved. The **relfrozenxid64** column is added to record the information.
intspnum	integer	Number of tablespaces that the interval partition belongs to
partkey	int2vector	Column number of the partition key
intervaltablespace	oidvector	Tablespace that the interval partition belongs to. Interval partitions fall in the tablespaces in the round-robin manner.
interval	text[]	Interval value of the interval partition
boundaries	text[]	Upper boundary of the range partition and interval partition
transit	text[]	Transit of the interval partition
reloptions	text[]	Storage property of a partition used for collecting online scale-out information. Same as **pg_class.reloptions**, it is expressed in a string in the format of keyword=value.
relfrozenxid64	xid	Frozen transaction ID
relminmxid	xid	Frozen multi-transaction ID

13.2.79 PG_PLTEMPLATE

PG_PLTEMPLATE records template information for procedural languages.

Table 13-80 PG_PLTEMPLATE columns

Name	Type	Description
tmplname	name	Name of the language for which this template is used
tmpltrusted	boolean	The value is **true** if the language is considered trusted.
tmpldbacreate	boolean	The value is **true** if the language is created by the owner of the database.
tmplhandler	text	Name of the call handler function
tmplinline	text	Name of the anonymous block handler (**NULL** if no name of the block handler exists)
tmplvalidator	text	Name of the verification function (**NULL** if no verification function is available)
tmpllibrary	text	Path of the shared library that implements languages
tmplacl	aclitem[]	Access permissions for template (not yet used)

13.2.80 PG_PROC

PG_PROC stores information about functions or stored procedures.

Table 13-81 PG_PROC columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified)
proname	name	Function name
pronamespace	oid	OID of the namespace that contains the function
proowner	oid	Owner of the function
prolang	oid	Implementation language or call interface of the function
procost	real	Estimated execution cost

Name	Type	Description
prorows	real	Estimated number of rows that are influenced
provariadic	oid	Data type of parameter element
protransform	regproc	Simplified call method for the function
proisagg	boolean	Whether the function is an aggregate function ● **t** (true): yes ● **f** (false): no
proiswindow	boolean	Whether the function is a window function ● **t** (true): yes ● **f** (false): no
prosecdef	boolean	Whether the function is a security definer (or a "setuid" function) ● **t** (true): yes ● **f** (false): no
proleakproof	boolean	This function has no side effect. If no leakproof treatment is provided for parameters, the function throws errors. ● **t** (true): There is no side effect. ● **f** (false): There are side effects.
proisstrict	boolean	The function returns null if any call parameter is null. In that case, the function is actually not called at all. Functions that are not "strict" must be prepared to process null inputs.
proretset	boolean	The function returns a set (multiple values of a specified data type).
provolatile	"char"	Whether the function's result depends only on its input parameters, or is affected by outside factors. ● **i**: for "immutable" functions, which always deliver the same result for the same inputs. ● **s**: for "stable" functions, whose results (for fixed inputs) do not change within a scan. ● **v**: for "volatile" functions, whose results may change at any time. Use **v** also for functions with side-effects, so that the engine cannot get optimized if volatile functions are called.
pronargs	smallint	Number of parameters
pronargdefaults	smallint	Number of parameters that have default values
prorettype	oid	Data type of return values

Name	Type	Description
proargtypes	oidvector	Array that stores the data types of function parameters. This array includes only input parameters (including **INOUT** parameters), and indicates the call signature (interface) of the function.
proallargtypes	oid[]	Array that contains the data types of function parameters. This array includes all parameter types (including **OUT** and **INOUT** parameters); however, if all the parameters are **IN** parameters, this column is null. Note that array subscripting is 1-based, whereas for historical reasons, **proargtypes** is subscripted from 0.
proargmodes	"char"[]	Array with the modes of the function parameters, encoded as follows: ● **i** indicates the IN parameter. ● **o** indicates the OUT parameter. ● **b** indicates the INOUT parameter. ● **v** indicates the VARIADIC parameter. If all the parameters are **IN** parameters, this column is null. Note that subscripts correspond to positions of **proallargtypes**, not **proargtypes**.
proargnames	text[]	Array that stores the names of the function parameters. Parameters without a name are set to empty strings in the array. If none of the parameters have a name, this column is null. Note that subscripts correspond to positions of **proallargtypes**, not **proargtypes**.
proargdefaults	pg_node_tree	Expression tree of the default value. This is the list of **pronargdefaults** elements.
prosrc	text	A definition that describes a function or stored procedure. In an interpreting language, it is the function source code, a link symbol, a file name, or any body content specified when a function or stored procedure is created, depending on how a language or call is used.
probin	text	Additional information about how to call the function. Again, the interpretation is language-specific.
proconfig	text[]	Function's local settings for run-time configuration variables.
proacl	aclitem[]	Access permissions. For details, see **GRANT** and **REVOKE**.

Name	Type	Description
prodefaultargpos	int2vector	Position of the input parameter of a function with a default value.
fencedmode	boolean	Execution mode of a function, indicating whether the function is executed in fence or not fence mode. If the execution mode is **fence,** the function is executed in the fork process that is reworked. In the C function created by the user, the default value of fencedmode is **true**, indicating the fence mode. For built-in functions in the system, the fencedmode field is set to **false**, indicating the not fence mode.
proshippable	boolean	Whether the function can be pushed down to database nodes. The default value is **false**. ● Functions of the IMMUTABLE type can always be pushed down to the database nodes. ● A STABLE or VOLATILE function can be pushed down to the database nodes only if SHIPPABLE is specified for it.
propackage	boolean	Whether the function supports overloading. The default value is **false**. ● **t** (true): supported. ● **f** (false): not supported.
prokind	"char"	Whether the object is a function or a stored procedure. ● **f** indicates that the object is a function. ● **p** indicates that the object is a stored procedure.
proargsrc	text	Describes the parameter input strings of functions or stored procedures that are compatible with Oracle syntax, including parameter comments. The default value is **NULL**.
proisprivate	boolean	Whether a function is a private function in the package. The default value is **false**.
propackageid	oid	OID of the package to which the function belongs. If the function is not in the package, the value is **0**.

Name	Type	Description
proargtypesext	oidvector_extend	Data type array used to store function parameters when there are a large number of function parameters. This array includes only input parameters (including **INOUT** parameters), and indicates the call signature (interface) of the function.
prodefaultargpo-sext	int2vector_extend	Position of the input parameter with a default value when the function has a large number of parameters.
allargtypes	oidvector	All stored procedure parameters (including input parameters, output parameters, and INOUT parameters), regardless of the parameter type.
allargtypesext	oidvector_extend	An array storing function parameters when there are a large number of function parameters. The array is a data type. All parameters (including input parameters, output parameters, and INOUT parameters) are included.

13.2.81 PG_PUBLICATION

PG_PUBLICATION contains all publications created in the current database.

Table 13-82 PG_PUBLICATION columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
pubname	name	Publication name.
pubowner	oid	Publication owner.
puballtables	boolean	If true, this publication automatically includes all tables in the database, including any tables that will be created in the future.
pubinsert	boolean	If true, copy the INSERT operation on tables in the publication.
pubupdate	boolean	If true, copy the UPDATE operation on tables in the publication.
pubdelete	boolean	If true, copy the DELETE operation on tables in the publication.

13.2.82 PG_PUBLICATION_REL

PG_PUBLICATION_REL contains mappings between tables and publications in the current database. This is a many-to-many mapping.

Table 13-83 PG_PUBLICATION_REL columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
prpubid	oid	Reference to a publication.
prrelid	oid	Reference to a table.

13.2.83 PG_RANGE

PG_RANGE records information about range types. Entries in **PG_TYPE** are excluded.

Table 13-84 PG_RANGE columns

Name	Type	Reference	Description
rngtypid	oid	**PG_TYPE**.oid	OID of the range type
rngsubtype	oid	**PG_TYPE**.oid	OID of the element type (subtype) of this range type
rngcollation	oid	**PG_COLLATION**.oid	OID of the collation used for range comparisons (**0** if none)
rngsubopc	oid	**PG_OPCLASS**.oid	OID of the subtype's operator class used for range comparisons
rngcanonical	regproc	**PG_PROC**.proname	Name of the function to convert a range value into canonical form (**0** if none)
rngsubdiff	regproc	**PG_PROC**.proname	Name of the function to return the difference between two element values as **double precision** (**0** if none)

rngsubopc (together with **rngcollation**, if the element type is collatable) determines the sort ordering used by the range type. **rngcanonical** is used when the element type is discrete.

13.2.84 PG_REPLICATION_ORIGIN

PG_REPLICATION_ORIGIN contains all created replication sources and is a globally shared table. That is, each node has only one copy, not one copy per database.

Table 13-85 PG_REPLICATION_ORIGIN columns

Name	Type	Description
roident	oid	Unique replication source identifier within a cluster.
roname	text	External user-defined replication source name.

13.2.85 PG_RESOURCE_POOL

PG_RESOURCE_POOL provides information about database resource pools.

Table 13-86 PG_RESOURCE_POOL columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified)
respool_name	name	Name of the resource pool
mem_percent	integer	Percentage of the memory configuration
cpu_affinity	bigint	Value of cores bound to the CPU
control_group	name	Name of the Cgroup where the resource pool is located
active_statements	integer	Maximum number of concurrent statements in the resource pool
max_dop	integer	Maximum scanning concurrency during data redistribution. This column is used for scaling.
memory_limit	name	Maximum memory of the resource pool
parentid	oid	OID of the parent resource pool
io_limits	integer	Upper limit of I/O operations per second. It is counted by ones for column storage and by 10 thousands for row storage.
io_priority	name	I/O priority set for jobs that consume many I/O resources. It takes effect when I/O usage reaches 90%.

Name	Type	Description
nodegroup	name	Name of the logical database to which the resource pool belongs
is_foreign	boolean	Whether the resource pool can be used for users outside the logical database. If it is set to **true**, the resource pool controls the resources of common users who do not belong to the current resource pool.
max_worker	integer	Concurrency in a table during data redistribution. This column is used only for scaling.

Note: **max_dop** and **max_worker** are used for scaling and are not applicable to the centralized deployment.

13.2.86 PG_REWRITE

PG_REWRITE records rewrite rules defined for tables and views.

Table 13-87 PG_REWRITE columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified)
rulename	name	Rule name
ev_class	oid	Name of the table that uses the rule
ev_attr	smallint	Column to which this rule applies (always **0** to indicate the entire table)
ev_type	"char"	Event type for the rule ● 1 = SELECT ● 2 = UPDATE ● 3 = INSERT ● 4 = DELETE
ev_enabled	"char"	Controls the mode in which the rule is triggered. ● **O**: The rule is triggered in origin and local modes. ● **D**: The rule is disabled. ● **R**: The rule is triggered in replica mode. ● **A**: The rule is always triggered.
is_instead	boolean	The value is **true** if the rule is of the **INSTEAD** type.

Name	Type	Description
ev_qual	pg_node_tree	Expression tree (in the form of a **nodeToString**() representation) for the rule's qualifying condition
ev_action	pg_node_tree	Query tree (in the form of a **nodeToString**() representation) for the rule's action

13.2.87 PG_RLSPOLICY

PG_RLSPOLICY records row-level access control policies.

Table 13-88 PG_RLSPOLICY columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified)
polname	name	Name of an access control policy
polrelid	oid	OID of the table object on which the row-level access control policy takes effect
polcmd	"char"	SQL operations affected by the row-level access control policy
polpermissive	boolean	Attribute of the row-level access control policy. **t** indicates an expression that uses the OR condition, and **f** indicates an expression that uses the AND condition.
polroles	oid[]	OID list of users affected by the row-level access control policy. If this parameter is not specified, all users are affected.
polqual	pg_node_tree	Expression of the row-level access control policy

13.2.88 PG_SECLABEL

PG_SECLABEL records security labels on database objects.

See also **PG_SHSECLABEL**, which provides a similar function for security labels of database objects that are shared within one database.

Table 13-89 PG_SECLABEL columns

Name	Type	Reference	Description
objoid	oid	Any OID column	OID of the object that this security label pertains to
classoid	oid	**PG_CLASS**.oid	OID of the system catalog where the object appears
objsubid	integer	-	Column number for a security label on a table column
provider	text	-	Label provider associated with the label
label	text	-	Security label applied to the object

13.2.89 PG_SHDEPEND

PG_SHDEPEND records the dependency between database objects and shared objects, such as roles. Based on this information, GaussDB can ensure that those objects are unreferenced before attempting to delete them.

See also **PG_DEPEND**, which provides a similar function for dependencies involving objects within a single database.

Unlike most system catalogs, **PG_SHDEPEND** is shared across all databases in the system. There is only one copy of **PG_SHDEPEND** in the entire database system, not one per database.

Table 13-90 PG_SHDEPEND columns

Name	Type	Reference	Description
dbid	oid	OID in **PG_DATABASE**	OID of the database where a dependent object is (**0** for a shared object)
classid	oid	OID in **PG_CLASS**	OID of the system catalog where a dependent object resides
objid	oid	Any OID column	OID of the dependent object
objsubid	integer	-	Column number for a table column (**objid** and **classid** refer to the table itself); The value is **0** for all other object types.
refclassid	oid	OID in **PG_CLASS**	OID of the system catalog where a referenced object is (must be a shared catalog)

Name	Type	Reference	Description
refobjid	oid	Any OID column	OID of the referenced object
deptype	"char"	-	Code segment defining the specific semantics of this dependency relationship. See the following for details.
objfile	text	-	Path of a user-defined function library file

In all cases, a **PG_SHDEPEND** entry indicates that the referenced object cannot be dropped without also dropping the dependent object. However, there are several subflavors identified by **deptype**:

- SHARED_DEPENDENCY_OWNER (o)

  The referenced object (which must be a role) is the owner of the dependent object.
- SHARED_DEPENDENCY_ACL (a)

  The referenced object (which must be a role) is mentioned in the access control list (ACL) of the dependent object. A **SHARED_DEPENDENCY_ACL** entry is not made for the owner of the object, since the owner will have a **SHARED_DEPENDENCY_OWNER** entry anyway.
- SHARED_DEPENDENCY_PIN (p)

  There is no dependent object. This type of entry is a signal that the system itself depends on the referenced object, and so that object must never be deleted. Entries of this type are created only by **initdb**. The columns for the dependent object contain zeroes.
- SHARED_DEPENDENCY_ DBPRIV(d)

  The referenced object (must be a role) has the ANY permission on the dependent object (the specified OID of the dependent object corresponds to a row in the **GS_DB_PRIVILEGE** system catalog).

13.2.90 PG_SHDESCRIPTION

PG_SHDESCRIPTION records optional comments for shared database objects. Descriptions can be manipulated with the **COMMENT** command and viewed with psql's **\d** commands.

See also **PG_DESCRIPTION**, which provides a similar function for descriptions involving objects within a single database.

Unlike most system catalogs, **PG_SHDESCRIPTION** is shared across all databases in the system. There is only one copy of **PG_SHDESCRIPTION** in the entire database system, not one per database.

Table 13-91 PG_SHDESCRIPTION columns

Name	Type	Reference	Description
objoid	oid	Any OID column	OID of the object that this description pertains to
classoid	oid	**PG_CLASS**.oid	OID of the system catalog where the object appears
description	text	-	Arbitrary text that serves as the description of the object

13.2.91 PG_SHSECLABEL

PG_SHSECLABEL records security labels on shared database objects. Security labels can be manipulated with the **SECURITY LABEL** command.

For an easier way to view security labels, see **PG_SECLABELS**.

See also **PG_SECLABEL**, which provides a similar function for security labels involving objects within a single database.

Unlike most system catalogs, **PG_SHSECLABEL** is shared across all databases in the system. There is only one copy of **PG_SHSECLABEL** in the GaussDB system, not one per database.

Table 13-92 PG_SHSECLABEL columns

Name	Type	Reference	Description
objoid	oid	Any OID column	OID of the object that this security label pertains to
classoid	oid	**PG_CLASS**.oid	OID of the system catalog where the object appears
provider	text	-	Label provider associated with the label
label	text	-	Security label applied to the object

13.2.92 PG_STATISTIC

PG_STATISTIC stores statistics about tables and index columns in a database. By default, only the system administrator can access the system catalog. Common users can access the system catalog only after being authorized.

Table 13-93 PG_STATISTIC columns

Name	Type	Description
starelid	oid	Table or index that the described column belongs to
starelkind	"char"	Type of an object
staattnum	smallint	Number of the described column in the table, starting from 1
stainherit	boolean	Whether to collect statistics for objects that have inheritance relationship
stanullfrac	real	Percentage of column entries that are null
stawidth	integer	Average stored width, in bytes, of non-null entries
stadistinct	real	Number of distinct, non-**NULL** data values in the column for database nodes ● A value greater than 0 is the actual number of distinct values. ● A value less than 0 is the negative of a multiplier for the number of rows in the table. (For example, **stadistinct=-0.5** indicates that values in a column appear twice on average.) ● The value **0** indicates that the number of distinct values is unknown.
stakindN	smallint	Code number stating that the type of statistics is stored in slot *N* of the **pg_statistic** row Value range: 1 to 5
staopN	oid	Operator used to generate the statistics stored in slot *N*. For example, a histogram slot shows the < operator that defines the sort order of the data. Value range: 1 to 5
stanumbersN	real[]	Numerical statistics of the appropriate type for slot *N*. The value is **NULL** if the slot does not involve numerical values. Value range: 1 to 5
stavaluesN	anyarray	Column data values of the appropriate type for slot *N*. The value is **NULL** if the slot type does not store any data values. Each array's element values are actually of the specific column's data type, so there is no way to define these columns' type more specifically than anyarray. Value range: 1 to 5

Name	Type	Description
stadndistinct	real	Number of unique non-null data values in the **dn1** column ● A value greater than 0 is the actual number of distinct values. ● A value less than 0 is the negative of a multiplier for the number of rows in the table. (For example, **stadistinct=-0.5** indicates that values in a column appear twice on average.) ● The value **0** indicates that the number of distinct values is unknown.
staextinfo	text	Information about extension statistics. This is reserved.

NOTICE

PG_STATISTIC stores sensitive information about statistical objects, such as MCVs. The system administrator and authorized users can access the **PG_STATISTIC** system catalog to query the sensitive information about the statistical objects.

13.2.93 PG_STATISTIC_EXT

PG_STATISTIC_EXT displays extended statistics of tables in a database, such as statistics of multiple columns. (The current feature is a lab feature. Contact Huawei technical support before using it.) Statistics of expressions will be supported later. You can specify the extended statistics to collect. This system catalog is accessible only to the system administrator.

Table 13-94 PG_STATISTIC_EXT columns

Name	Type	Description
starelid	oid	Table or index that the described column belongs to
starelkind	char	Type of the object to which a table belongs. **c** indicates an ordinary table, and **p** indicates a partitioned table.
stainherit	boolean	Whether to collect statistics for objects that have inheritance relationship
stanullfrac	real	Percentage of column entries that are null
stawidth	integer	Average stored width, in bytes, of non-null entries

Name	Type	Description
stadistinct	real	Number of distinct, non-**NULL** data values in the column for database nodes ● A value greater than 0 is the actual number of distinct values. ● A value less than 0 is the negative of a multiplier for the number of rows in the table. (For example, **stadistinct=-0.5** indicates that values in a column appear twice on average.) ● The value **0** indicates that the number of distinct values is unknown.
stadndistinct	real	Number of unique non-null data values in the **dn1** column ● A value greater than 0 is the actual number of distinct values. ● A value less than 0 is the negative of a multiplier for the number of rows in the table. (For example, **stadistinct=-0.5** indicates that values in a column appear twice on average.) ● The value **0** indicates that the number of distinct values is unknown.
stakindN	smallint	Code number stating that the type of statistics is stored in slot *N* of the **pg_statistic** row Value range: 1 to 5
staopN	oid	Operator used to generate the statistics stored in slot *N*. For example, a histogram slot shows the < operator that defines the sort order of the data. Value range: 1 to 5
stakey	int2vector	Array of a column ID
stanumbersN	real[]	Numerical statistics of the appropriate type for slot *N*. The value is **NULL** if the slot does not involve numerical values. Value range: 1 to 5
stavaluesN	anyarray	Column data values of the appropriate type for slot *N*. The value is **NULL** if the slot type does not store any data values. Each array's element values are actually of the specific column's data type so there is no way to define these columns' type more specifically than anyarray. Value range: 1 to 5
staexprs	pg_node_tree	Expression corresponding to the extended statistics information.

NOTICE

PG_STATISTIC_EXT stores sensitive information about statistical objects, such as MCVs. The system administrator and authorized users can access the **PG_STATISTIC_EXT** system catalog to query the sensitive information about the statistical objects.

13.2.94 PG_SUBSCRIPTION

PG_SUBSCRIPTION contains all existing logical replication subscriptions. This system catalog is accessible only to system administrators.

Unlike most system catalogs, **PG_SUBSCRIPTION** is shared across all databases in a database instance. Each database instance has only one copy, not one copy per database.

Table 13-95 PG_SUBSCRIPTION columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
subdbid	oid	OID of the database where the subscription is located.
subname	name	Subscription name.
subowner	oid	Subscription owner.
subenabled	boolean	If true, the subscription is enabled and should be replicated.
subconninfo	text	Information about the connection to the publishing database.
subslotname	name	Name of the replication slot in the publishing database. If this parameter is left empty, the value is **NONE**.
subsynccommit	text	Value of **synchronous_commit** of the subscription worker.
subpublications	text[]	Array containing names of the subscribed publications. These are referenced publications on the publisher server.

13.2.95 PG_SYNONYM

PG_SYNONYM records the mapping between synonym object names and other database object names.

Table 13-96 PG_SYNONYM columns

Name	Type	Description
oid	oid	Database object ID
synname	name	Synonym name
synnamespace	oid	OID of the namespace that contains a synonym
synowner	oid	Owner of a synonym, usually the OID of the user who created it
synobjschema	name	Schema name specified by an associated object
synobjname	name	Name of an associated object

13.2.96 PG_TABLESPACE

PG_TABLESPACE records tablespace information.

Table 13-97 PG_TABLESPACE columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified)
spcname	name	Tablespace name
spcowner	oid	Owner of the tablespace, usually the user who created it
spcacl	aclitem[]	Access permissions. For details, see **GRANT** and **REVOKE**.
spcoptions	text[]	Options of the tablespace
spcmaxsize	text	Maximum size of the available disk space, in bytes
relative	boolean	Whether the storage path specified by the tablespace is a relative path

13.2.97 PG_TRIGGER

PG_TRIGGER records trigger information.

Table 13-98 PG_TRIGGER columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified)
tgrelid	oid	OID of the table where the trigger is located
tgname	name	Trigger name
tgfoid	oid	Function to be invoked by the trigger
tgtype	smallint	Trigger type
tgenabled	"char"	**O**: The trigger is triggered in origin or local mode. **D**: The trigger is disabled. **R**: The trigger is triggered in replica mode. **A**: The trigger is always triggered.
tgisinternal	boolean	Internal trigger ID. If the value is **true**, it indicates an internal trigger.
tgconstrrelid	oid	Table referenced by the integrity constraint
tgconstrindid	oid	Index of the integrity constraint
tgconstraint	oid	OID of the constraint trigger in **PG_CONSTRAINT**
tgdeferrable	boolean	Whether the constraint trigger is of the DEFERRABLE type
tginitdeferred	boolean	Whether the trigger is of the INITIALLY DEFERRED type
tgnargs	smallint	Number of input parameters of the trigger function
tgattr	int2vector	Column ID specified by the trigger. If no column is specified, an empty array is used.
tgargs	bytea	Parameter transferred to the trigger
tgqual	pg_node_tree	WHEN condition of the trigger (**NULL** if the WHEN condition does not exist)
tgowner	oid	Trigger owner

13.2.98 PG_TS_CONFIG

PG_TS_CONFIG contains entries representing text search configurations. A configuration specifies a particular text search parser and a list of dictionaries to use for each of the parser's output token types.

The parser is shown in the **PG_TS_CONFIG** entry, but the token-to-dictionary mapping is defined by subsidiary entries in **PG_TS_CONFIG_MAP**.

Table 13-99 PG_TS_CONFIG columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)
cfgname	name	-	Text search configuration name
cfgnamespace	oid	**PG_NAMESPACE**.oid	OID of the namespace that contains the configuration
cfgowner	oid	**PG_AUTHID**.oid	Owner of the configuration
cfgparser	oid	**PG_TS_PARSER**.oid	OID of the text search parser for this configuration
cfoptions	text[]	-	Configuration options

13.2.99 PG_TS_CONFIG_MAP

PG_TS_CONFIG_MAP contains entries showing which text search dictionaries should be consulted, and in what order, for each output token type of each text search configuration's parser.

Table 13-100 PG_TS_CONFIG_MAP columns

Name	Type	Reference	Description
mapcfg	oid	OID in **PG_TS_CONFIG**	OID of the **PG_TS_CONFIG** entry owning this map entry
maptokentype	integer	-	Token type generated by the configuration's parser
mapseqno	integer	-	Sequence number of a token type when the values of **mapcfg** or **maptokentype** are the same
mapdict	oid	OID in **PG_TS_DICT**	OID of the text search dictionary to consult

13.2.100 PG_TS_DICT

PG_TS_DICT contains entries that define text search dictionaries. A dictionary depends on a text search template, which specifies all the implementation functions needed; the dictionary itself provides values for the user-settable parameters supported by the template.

This division of labor allows dictionaries to be created by unprivileged users. The parameters are specified by a text string **dictinitoption**, whose format and meaning vary depending on the template.

Table 13-101 PG_TS_DICT columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)
dictname	name	-	Text search dictionary name
dictnamespace	oid	**PG_NAMESPACE**.oid	OID of the namespace that contains the dictionary
dictowner	oid	**PG_AUTHID**.oid	Owner of the dictionary
dicttemplate	oid	**PG_TS_TEMPLATE**.oid	OID of the text search template for the dictionary
dictinitoption	text	-	Initialization option string for the template

13.2.101 PG_TS_PARSER

PG_TS_PARSER contains entries defining text search parsers. A parser is responsible for splitting input text into lexemes and assigning a token type to each lexeme. The new parser must be created by the database system administrator.

Table 13-102 PG_TS_PARSER columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)
prsname	name	-	Text search parser name
prsnamespace	oid	**PG_NAMESPACE**.oid	OID of the namespace that contains the parser
prsstart	regproc	**PG_PROC**.proname	Name of the parser's startup function
prstoken	regproc	**PG_PROC**.proname	Name of the parser's next-token function

Name	Type	Reference	Description
prsend	regproc	PG_PROC.proname	Name of the parser's shutdown function
prsheadline	regproc	PG_PROC.proname	Name of the parser's headline function
prslextype	regproc	PG_PROC.proname	Name of the parser's lextype function

13.2.102 PG_TS_TEMPLATE

PG_TS_TEMPLATE contains entries defining text search templates. A template provides a framework for text search dictionaries. Since a template must be implemented by C-language-level functions, templates can be created only by database administrators.

Table 13-103 PG_TS_TEMPLATE columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)
tmplname	name	-	Text search template name
tmplnamespace	oid	PG_NAMESPACE.oid	OID of the namespace that contains the template
tmplinit	regproc	PG_PROC.proname	Name of the template's initialization function
tmpllexize	regproc	PG_PROC.proname	Name of the template's lexize function

13.2.103 PG_TYPE

PG_TYPE stores information about data types.

Table 13-104 PG_TYPE columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
typname	name	Data type name

Name	Type	Description
typnamespace	oid	OID of the namespace that contains the type
typowner	oid	Owner of the type
typlen	smallint	Number of bytes in the internal representation of the type for a fixed-size type. It is a negative number for a variable-length type. ● The value **–1** indicates a "varlena" type (one that has a length word). ● The value **–2** indicates a null-terminated C string.
typbyval	boolean	Specifies whether to pass a value (**true**) or a reference (**false**) when a value of this type is passed internally. **typbyval** is set to **false** if the value of **typlen** is not **1**, **2**, **4**, or **8**, because values of this type are always passed by reference of this column. **typbyval** can be **false** even if the **typlen** is passed by a parameter of this column.
typtype	"char"	● **b**: base type. ● **c**: composite type (for example, a table's row type) ● **d**: domain ● **p**: pseudo ● **o**: set type For details, see **typrelid** and **typbasetype**.
typcategory	"char"	**typcategory** is an arbitrary classification of data types that is used by the parser to determine which implicit casts should be preferred.
typispreferred	boolean	The value is **true** if conversion is performed when data meets conversion rules specified by **typcategory**.
typisdefined	boolean	Whether a type has been defined. It is **true** if the type is defined, and **false** if this is a placeholder entry for a not-yet-defined type. When it is **false**, nothing except the type name, namespace, and OID can be relied on.
typdelim	"char"	Character that separates two values of this type when parsing an array input. Note that the delimiter is associated with the array element data type, not the array data type.

Name	Type	Description
typrelid	oid	If this is a composite type (see **typtype**), then this column points to the **PG_CLASS** entry that defines the corresponding table. For a free-standing composite type, the **PG_CLASS** entry does not represent a table, but it is required for the type's **PG_ATTRIBUTE** entries to link to. It is **0** for non-composite type.
typelem	oid	If **typelem** is not **0**, it identifies another row in **PG_TYPE**. The current type can be described as an array yielding values of type **typelem**. A "true" array type has a variable length (**typlen** = **–1**), but some fixed-length types (**typlen** > **0**) also have non-zero **typelem**, for example **name** and **point**. If a fixed-length type has a **typelem**, its internal representation must be a number of values of the **typelem** data type with no other data. Variable-length array types have a header defined by the array subroutines.
typarray	oid	If the value is not **0**, the corresponding type record is available in **PG_TYPE**.
typinput	regproc	Input conversion function (text format)
typoutput	regproc	Output conversion function (text format)
typreceive	regproc	Input conversion function (binary format); **0** for non-input conversion function
typsend	regproc	output conversion function (binary format); **0** for non-output conversion function
typmodin	regproc	Type modifier input function; **0** if the type does not support modifiers
typmodout	regproc	Type modifier output function; **0** if the type does not support modifiers
typanalyze	regproc	Custom **ANALYZE** function; **0** if the standard function is used

Name	Type	Description
typalign	"char"	Alignment required when storing a value of this type. It applies to storage on disks as well as most representations of the value inside GaussDB. When multiple values are stored consecutively, such as in the representation of a complete row on disk, padding is inserted before a data of this type so that it begins on the specified boundary. The alignment reference is the beginning of the first datum in the sequence. Possible values are: ● **c**: char alignment, that is, no alignment needed ● **s**: short alignment (2 bytes on most machines) ● **i**: integer alignment (4 bytes on most machines) ● **d**: double alignment (8 bytes on many machines, but by no means all) **NOTICE** For types used in system tables, the size and alignment defined in **PG_TYPE** must agree with the way that the compiler lays out the column in a structure representing a table row.
typstorage	"char"	**typstorage** tells for varlena types (those with **typlen = –1**) if the type is prepared for toasting and what the default strategy for attributes of this type should be. Possible values are: ● **p**: Values are always stored plain. ● **e**: Values can be stored in a secondary relationship (if the relation has one, see **pg_class.reltoastrelid**). ● **m**: Values can be stored compressed inline. ● **x**: Values can be stored compressed inline or stored in secondary storage. **NOTICE** **m** domains can also be moved out to secondary storage, but only as a last resort (**e** and **x** domains are moved first).
typnotnull	boolean	Whether the type has a NOTNULL constraint. Currently, it is used for domains only.
typbasetype	oid	If this is a domain (see **typtype**), then **typbasetype** identifies the type that this one is based on. The value is **0** if this type is not a derived type.
typtypmod	integer	Records the **typtypmod** to be applied to domains' base types by domains (the value is **–1** if the base type does not use **typmod**). This is **–1** if this type is not a domain.

Name	Type	Description
typndims	integer	Number of array dimensions for a domain that is an array (**typbasetype** is an array type; the domain's **typelem** matches the base type's **typelem**). This is **0** for types other than domains over array types.
typcollation	oid	Sequence rule for specified types. For details about the value, see the system catalog in PG_COLLATION. (**0** if sequencing is not supported)
typdefaultbin	pg_node_tree	**nodeToString()** representation of a default expression for the type if the value is non-null. Currently, this column is only used for domains.
typdefault	text	The value is **NULL** if a type has no associated default value. ● If **typdefaultbin** is not **NULL**, **typdefault** must contain a default expression represented by **typdefaultbin**. ● If **typdefaultbin** is **NULL** and **typdefault** is not, then **typdefault** is the external representation of the type's default value, which can be fed to the type's input converter to produce a constant.
typacl	aclitem[]	Access permission

13.2.104 PG_USER_MAPPING

PG_USER_MAPPING records mappings from local users to remote.

This system catalog is accessible only to system administrators. Common users can query the **PG_USER_MAPPINGS** view.

Table 13-105 PG_USER_MAPPING columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)
umuser	oid	**PG_AUTHID**.oid	OID of the local role being mapped (**0** if the user mapping is public)
umserver	oid	**PG_FOREIGN_SERVER**.oid	OID of the foreign server that contains the mapping
umoptions	text[]	-	User mapping specific options, expressed in a string in the format of keyword=value

13.2.105 PG_USER_STATUS

PG_USER_STATUS provides the states of users who access the database. This system catalog is accessible only to system administrators.

Table 13-106 PG_USER_STATUS columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified)
roloid	oid	ID of the role
failcount	integer	Number of failed attempts
locktime	timestamp with time zone	Time at which the role is locked
rolstatus	smallint	Role state ● **0**: normal ● **1**: The role is locked for a specific period of time because the failed login attempts exceed the threshold. ● **2**: The role is locked by the administrator.
permspace	bigint	Size of the permanent table storage space used by the role
tempspace	bigint	Size of the temporary table storage space used by the role
passwordexpired	smallint	Whether a password is valid. ● **0**: The password is valid. ● **1**: The password is invalid.

13.2.106 PG_WORKLOAD_GROUP

PG_WORKLOAD_GROUP provides workload group information in the database.

Table 13-107 PG_WORKLOAD_GROUP columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified)
workload_gpname	name	Workload group name
respool_oid	oid	ID bound to the resource pool

Name	Type	Description
act_statements	integer	Maximum number of active statements in the workload group

13.2.107 PGXC_CLASS

PGXC_CLASS records the replicated or distributed information for each table. The **PGXC_CLASS** system catalog has specific meanings only in distributed scenarios. Centralized systems can query only the table definition.

Table 13-108 PGXC_CLASS columns

Name	Type	Description
pcrelid	oid	Table OID
pclocatortype	"char"	Locator type ● **H**: Hash ● **G**: Range ● **L**: List ● **M**: Modulo ● **N**: Round Robin ● **R**: Replication
pchashalgorithm	smallint	Distributed tuple using the hash algorithm
pchashbuckets	smallint	Value of a hash container
pgroup	name	Name of the node
redistributed	"char"	Indicates that a table has been redistributed.
redis_order	integer	Redistribution sequence. Tables whose values are **0** will not be redistributed in this round of redistribution.
pcattnum	int2vector	Column number used as a distributed key
nodeoids	oidvector_extend	List of distributed table node OIDs
options	text	Extension status information. This is a reserved column in the system.

13.2.108 PGXC_GROUP

PGXC_GROUP records information about node groups. The **PGXC_GROUP** system catalog has specific meanings only in distributed scenarios. Centralized systems can query only the table definition.

Table 13-109 PGXC_GROUP columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified)
group_name	name	Node group name
in_redistribution	"char"	Whether redistribution is required. The value must be one of the following: ● **n**: The node group is not redistributed. ● **y**: The source node group is in redistribution. ● **t**: The destination node group is in redistribution.
group_members	oidvector_extend	Node OID list of the node group
group_buckets	text	Distributed data bucket group
is_installation	boolean	Specifies whether to install a sub-database instance. ● **t** (true): yes ● **f** (false): no
group_acl	aclitem[]	Access permission
group_kind	"char"	Node group type. The value must be one of the following: ● **i**: installation node group ● **n**: node group in a common non-logical database instance ● **v**: node group in a logical database instance ● **e**: elastic database instance
group_parent	oid	For a child node group, this field indicates the OID of the parent node group. For a parent node group, this field is left blank.

13.2.109 PGXC_NODE

The **PGXC_NODE** system catalog stores information about database instance nodes. This system catalog has specific meanings only in distributed scenarios. Centralized systems can query only the table definition.

Table 13-110 PGXC_NODE columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified)
node_name	name	Node name
node_type	"char"	Node type ● **C**: CN ● **D**: DN ● **S**: standby node
node_port	integer	Port number of the node
node_host	name	Host name or IP address of a node. (If a virtual IP address is configured, its value is a virtual IP address.)
node_port1	integer	Port number of a replication node
node_host1	name	Host name or IP address of a replication node. (If a virtual IP address is configured, its value is a virtual IP address.)
hostis_primary	boolean	Whether a primary/standby switchover occurs on the current node ● **t** (true): yes ● **f** (false): no
nodeis_primary	boolean	Whether the current node is preferred to execute non-query operations in the **replication** table ● **t** (true): yes ● **f** (false): no
nodeis_preferred	boolean	Whether the current node is preferred to execute queries in the **replication** table ● **t** (true): yes ● **f** (false): no
node_id	integer	Node identifier. The value is obtain by calculating the value of **node_name** using the hash function.
sctp_port	integer	Port used by the TCP proxy communication library or SCTP communication library of the primary node to listen on the data channel. (Due to specification changes, the current version no longer supports the current feature. Do not use this feature.)

Name	Type	Description
control_port	integer	Port used by the TCP proxy communications library of the primary node to listen on the control channel
sctp_port1	integer	Port used by the TCP proxy communication library or SCTP communication library (Due to specification changes, the current version no longer supports the current feature. Do not use this feature.) of the standby node to listen on the data channel
control_port1	integer	Port used by the TCP proxy communications library of the standby node to listen on the control channel
nodeis_central	boolean	Whether the current node is a CN. It is invalid for DNs. ● **t** (true): yes ● **f** (false): no
nodeis_active	boolean	Whether the current node is normal. It is invalid for DNs. ● **t** (true): yes ● **f** (false): no

13.2.110 PGXC_SLICE

PGXC_SLICE is a system catalog created for recording range distribution and list distribution details. Currently, range interval cannot be used to automatically scale out shards. It is reserved in the system catalog.

This system catalog has specific meanings only in distributed scenarios. Centralized systems can query only the table definition.

Table 13-111 PGXC_SLICE columns

Name	Type	Description
relname	name	Table name or shard name, which is distinguished by **type**.
type	"char"	When the value is **t**, **relname** indicates the table name. When the value is **s**, **relname** indicates the shard name.
strategy	"char"	**r**: range distribution table **l**: list distribution table This value will be extended for subsequent interval shards.

Name	Type	Description
relid	oid	OID of the distribution table to which the tuple belongs.
referenceoid	oid	OID of the referenced distribution table, which is used for slice reference table creation syntax.
sindex	integer	Position of the current boundary in a shard when the table is a list distribution table.
interval	text[]	Reserved column
transitboundary	text[]	Reserved column
transitno	integer	Reserved column
nodeoid	oid	When **relname** is set to a shard name, **nodeoid** indicates the OID of the DN where the shard data is stored.
boundaries	text[]	When **relname** is set to a shard name, this parameter indicates the boundary value of the shard.
specified	boolean	Whether the DN corresponding to the current segment is explicitly specified in the DDL.
sliceorder	integer	User-defined shard sequence.

13.2.111 PLAN_TABLE_DATA

PLAN_TABLE_DATA stores plan information collected by **EXPLAIN PLAN**. Different from the **PLAN_TABLE** view, the system catalog **PLAN_TABLE_DATA** stores **EXPLAIN PLAN** information collected by all sessions and users.

Table 13-112 PLAN_TABLE_DATA columns

Name	Type	Description
session_id	text	Session that inserts the data. Its value consists of a service thread start timestamp and a service thread ID. Values are constrained by **NOT NULL**.
user_id	oid	User who inserts the data. Values are constrained by **NOT NULL**.
statement_id	varchar2(30)	Query tag specified by a user

Name	Type	Description
plan_id	bigint	Query ID The ID is automatically generated in the plan generation phase and is used by kernel engineers for debugging.
id	integer	Node ID in a plan
operation	varchar2(30)	Operation description
options	varchar2(255)	Operation action
object_name	name	Name of an operated object. It is defined by users.
object_type	varchar2(30)	Object type
object_owner	name	Schema to which the object belongs. It is defined by users.
projection	varchar2(4000)	Returned column information
cost	double precision	Execution cost estimated by the optimizer for an operator
cardinality	double precision	Number of rows estimated by the optimizer for an operator

NOTE

- **PLAN_TABLE_DATA** records data of all users and sessions on the current node. Only administrators can access all the data. Common users can view their own data in the **PLAN_TABLE** view.
- Data is automatically inserted into **PLAN_TABLE_DATA** after **EXPLAIN PLAN** is executed. Therefore, do not manually insert data into or update data in **PLAN_TABLE_DATA**. Otherwise, data in **PLAN_TABLE_DATA** may be disordered. To delete data from **PLAN_TABLE_DATA**, you are advised to use the **PLAN_TABLE** view.
- Information in the **statement_id**, **object_name**, **object_owner**, and **projection** columns is stored in letter cases specified by users and information in other columns is stored in uppercase.

13.2.112 STATEMENT_HISTORY

STATEMENT_HISTORY displays information about execution statements on the current node. To query this system catalog, you must have the **sysadmin** permission. The result can be queried only in the system database but cannot be queried in the user database.

The constraints on the query of this system catalog are as follows:

- Data must be queried in the Postgres database. No data exists in other databases.

- This system catalog is controlled by **track_stmt_stat_level**. The default value is **OFF,L0**, where the first part controls full SQL statements, and the second part controls slow SQL statements. For details about the record level of each field, see the following table.
- For slow SQL statements, if the value of **track_stmt_stat_level** is not **OFF** and the SQL execution time exceeds the value of **log_min_duration_statement**, the SQL statement is recorded as a slow SQL statement.

Table 13-113 STATEMENT_HISTORY columns

Name	Type	Description	Record Level
db_name	name	Database name.	L0
schema_name	name	Schema name.	L0
origin_node	integer	Node name.	L0
user_name	name	Username.	L0
application_name	text	Name of the application that sends a request.	L0
client_addr	text	IP address of the client that sends a request.	L0
client_port	integer	Port number of the client that sends a request.	L0
unique_query_id	bigint	ID of the normalized SQL statement.	L0
debug_query_id	bigint	ID of the unique SQL statement.	L0
query	text	Normalized SQL statement.	L0
start_time	timestamp with time zone	Time when a statement starts.	L0
finish_time	timestamp with time zone	Time when a statement ends.	L0
slow_sql_threshold	bigint	Standard for slow SQL statement execution.	L0
transaction_id	bigint	Transaction ID.	L0
thread_id	bigint	ID of an execution thread.	L0
session_id	bigint	Session ID of a user.	L0

Name	Type	Description	Record Level
n_soft_parse	bigint	Number of soft parsing times. The value of **n_soft_parse** plus the value of **n_hard_parse** may be greater than the value of **n_calls** because the number of subqueries is not counted in the value of **n_calls**.	L0
n_hard_parse	bigint	Number of hard parsing times. The value of **n_soft_parse** plus the value of **n_hard_parse** may be greater than the value of **n_calls** because the number of subqueries is not counted in the value of **n_calls**.	L0
query_plan	text	Statement execution plan.	L1
n_returned_rows	bigint	Number of rows in the result set returned by the **SELECT** statement.	L0
n_tuples_fetched	bigint	Number of rows randomly scanned.	L0
n_tuples_returned	bigint	Number of rows sequentially scanned.	L0
n_tuples_inserted	bigint	Number of rows inserted.	L0
n_tuples_updated	bigint	Number of rows updated.	L0
n_tuples_deleted	bigint	Number of rows deleted.	L0
n_blocks_fetched	bigint	Number of buffer block access times.	L0
n_blocks_hit	bigint	Number of buffer block hits.	L0
db_time	bigint	Valid DB time, which is accumulated if multiple threads are involved (unit: μs).	L0
cpu_time	bigint	CPU time (unit: μs).	L0
execution_time	bigint	Execution time in the executor (unit: μs).	L0
parse_time	bigint	SQL parsing time (unit: μs).	L0
plan_time	bigint	SQL plan generation time (unit: μs).	L0
rewrite_time	bigint	SQL rewriting time (unit: μs).	L0

Name	Type	Description	Record Level
pl_execution_time	bigint	Execution time of PL/pgSQL (unit: μs).	L0
pl_compilation_time	bigint	Compilation time of PL/pgSQL (unit: μs).	L0
data_io_time	bigint	I/O time (unit: μs).	L0
net_send_info	text	Network status of messages sent through a physical connection, including the time (unit: μs), number of calls, and throughput (unit: byte). This can be used to analyze the network overhead of SQL statements in a distributed system and is not supported in standalone system. Example: {"time":xxx, "n_calls":xxx, "size":xxx}.	L0
net_recv_info	text	Network status of messages received through a physical connection, including the time (unit: μs), number of calls, and throughput (unit: byte). This column can be used to analyze the network overhead of SQL in a distributed system. This column is not supported in a standalone mode. Example: {"time":xxx, "n_calls":xxx, "size":xxx}.	L0
net_stream_send_info	text	Network status of messages sent through a logical connection, including the time (unit: μs), number of calls, and throughput (unit: byte). This column can be used to analyze the network overhead of SQL in a distributed system. This column is not supported in a standalone mode. Example: {"time":xxx, "n_calls":xxx, "size":xxx}.	L0

Name	Type	Description	Record Level
net_stream_recv_info	text	Network status of messages received through a logical connection, including the time (unit: μs), number of calls, and throughput (unit: byte). This column can be used to analyze the network overhead of SQL in a distributed system. This column is not supported in a standalone mode. Example: {"time":xxx, "n_calls":xxx, "size":xxx}.	L0
lock_count	bigint	Number of locks.	L0
lock_time	bigint	Time required for locking.	L1
lock_wait_count	bigint	Number of lock waits.	L0
lock_wait_time	bigint	Time required for lock waiting.	L1
lock_max_count	bigint	Maximum number of locks.	L0
lwlock_count	bigint	Number of lightweight locks (reserved).	L0
lwlock_wait_count	bigint	Number of lightweight lock waits.	L0
lwlock_time	bigint	Time required for lightweight locking (reserved).	L1
lwlock_wait_time	bigint	Time required for lightweight lock waiting.	L1

Name	Type	Description	Record Level
details	bytea	List of statement lock events, which are recorded in time sequence. The number of records is affected by the **track_stmt_details_size** parameter. This field is in binary format and needs to be read by using the parsing function **pg_catalog.statement_detail_decode**. For details, see **Table 11-55**. Events include: ● Start locking. ● Complete locking. ● Start lock waiting. ● Complete lock waiting. ● Start unlocking. ● Complete unlocking. ● Start lightweight lock waiting. ● Complete lightweight lock waiting.	L2
is_slow_sql	boolean	Whether the SQL statement is a slow SQL statement. ● **t** (true): yes ● **f** (false): no	L0
trace_id	text	Driver-specific trace ID, which is associated with an application request.	L0

Name	Type	Description	Record Level
advise	text	Risks which may cause slow SQL statements. (Multiple risks may exist at the same time.) ● Cast Function Cause Index Miss. Index matching may fail due to implicit conversion. ● Limit too much rows. The SQL statement execution may slow down due to a large **limit** value. ● Proleakproof of function is false. The **proleakproof** of the function is set to **false**. In this case, the function does not use statistics when generating a plan due to data leakage risks. As a result, the accuracy of the generated plan is affected and the SQL statement execution may slow down.	L0

13.2.113 STREAMING_STREAM

STREAMING_STREAM records the metadata of all STREAM objects.

Table 13-114 STREAMING_STREAM columns

Name	Type	Description
relid	oid	STREAM object ID.
queries	bytea	Bitmap mapping of the CONTVIEW corresponding to the STREAM.

13.2.114 STREAMING_CONT_QUERY

STREAMING_CONT_QUERY records the metadata of all CONTVIEW objects.

Table 13-115 STREAMING_CONT_QUERY columns

Name	Type	Description
id	integer	Unique identifier of the CONTVIEW object.

Name	Type	Description
type	"char"	CONTVIEW type. ● **c** indicates that the CONTVIEW is based on the column-store model. ● **r** indicates that the CONTVIEW is based on the row-store model. ● **p** indicates that the CONTVIEW is based on the partitioned column-store model.
relid	oid	CONTVIEW object ID.
defrelid	oid	ID of the continuous computing rule view corresponding to CONTVIEW.
active	boolean	Determines whether the CONTVIEW is in the continuous computing state. ● **t** (true): yes ● **f** (false): no
streamrelid	oid	ID of STREAM corresponding to CONTVIEW.
matrelid	oid	ID of the materialized table corresponding to CONTVIEW.
lookupidxid	oid	ID of GROUP LOOK UP INDEX corresponding to CONTVIEW. This column is for internal use and is available only in row-store tables.
step_factor	smallint	CONTVIEW step mode. The main values are **0** (no overlapping window) and **1** (sliding window, with one step).
ttl	integer	Value of **ttl_interval** set by CONTVIEW.
ttl_attno	smallint	Number of a time column corresponding to the TTL function set by CONTVIEW.
dictrelid	oid	ID of the dictionary table corresponding to CONTVIEW.
grpnum	smallint	Number of dimension columns in the CONTVIEW continuous computing rule. This column is for internal use.
grpidx	int2vector	Index of the dimension column in TARGET LIST in the CONTVIEW continuous computing rule. This column is for internal use.

13.2.115 STREAMING_REAPER_STATUS

STREAMING_REAPER_STATUS records the status information about the reaper thread of the streaming engine. (Due to specification changes, the current version no longer supports the current feature. Do not use this feature.)

Table 13-116 STREAMING_REAPER_STATUS columns

Name	Type	Description
id	integer	Unique identifier of the CONTVIEW object.
contquery_name	name	Name of the CONTVIEW object.
gather_interval	text	Value of **gather_interval** (time parameter for automatically aggregating historical data before a specific time) set for the CONTVIEW object.
gather_completion_time	text	Time when the latest GATHER (historical data aggregation) operation on the CONTVIEW object is completed.

13.3 System Views

13.3.1 ADM_COL_COMMENTS

ADM_COL_COMMENTS displays information about table column comments in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both **PG_CATALOG** and **SYS schema**.

Table 13-117 ADM_COL_COMMENTS columns

Name	Type	Description
owner	character varying(64)	Table owner
table_name	character varying(64)	Table name
column_name	character varying(64)	Column name
comments	text	Comments

13.3.2 ADM_CONSTRAINTS

ADM_CONSTRAINTS displays information about table constraints in database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both **PG_CATALOG** and **SYS schema**.

Table 13-118 ADM_CONSTRAINTS columns

Name	Type	Description
owner	character varying(64)	Constraint creator
constraint_name	character varying(64)	Constraint name
constraint_type	text	Constraint type ● **c**: check constraint ● **f**: foreign key constraint ● **p**: primary key constraint ● **u**: unique constraint
table_name	character varying(64)	Name of a constraint-related table
index_owner	character varying(64)	Owner of a constraint-related index (only for the unique constraint and primary key constraint)
index_name	character varying(64)	Name of a constraint-related index (only for the unique constraint and primary key constraint)

13.3.3 ADM_CONS_COLUMNS

ADM_CONS_COLUMNS displays information about constraint columns in database tables. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both **PG_CATALOG** and **SYS schema**.

Table 13-119 ADM_CONS_COLUMNS columns

Name	Type	Description
owner	character varying(64)	Constraint creator
column_name	character varying(64)	Name of a constraint-related column
constraint_name	character varying(64)	Constraint name
position	smallint	Position of a column in a table

Name	Type	Description
table_name	character varying(64)	Name of a constraint-related table

13.3.4 ADM_DATA_FILES

ADM_DATA_FILES displays the description of database files. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in the **PG_CATALOG** and **SYS** schemas.

Table 13-120 ADM_DATA_FILES columns

Name	Type	Description
tablespace_name	name	Name of the tablespace to which a file belongs
bytes	double precision	Length of a file, in bytes

13.3.5 ADM_HIST_SNAPSHOT

ADM_HIST_SNAPSHOT displays the index information and start time of WDR snapshots stored in the current system. By default, only the system administrator can access this view. Common users can access the view only after being authorized.

Table 13-121 ADM_HIST_SNAPSHOT columns

Name	Type	Description
SNAP_ID	bigint	WDR snapshot ID
BEGIN_INTERVAL_TIME	timestamp	Start time of a WDR snapshot

13.3.6 ADM_HIST_SQL_PLAN

ADM_HIST_SQL_PLAN displays plan information collected by the current user by running the **EXPLAIN PLAN** statement. By default, only the system administrator can access this view. Common users can access the view only after being authorized.

Table 13-122 ADM_HIST_SQL_PLAN columns

Name	Type	Description
SQL_ID	character varying(30)	Session that inserts the data. Its value consists of a service thread start timestamp and a service thread ID. Values are constrained by **NOT NULL**.
PLAN_HASH_VALUE	bigint	Query ID.
OPERATION	character varying(30)	Operation description.
OPTIONS	character varying(255)	Operation action.
OBJECT_NAME	name	Name of an operated object. It is defined by users.

13.3.7 ADM_HIST_SQLSTAT

ADM_HIST_SQLSTAT displays information about statements executed on the current node. By default, only the system administrator can access this view. Common users can access the view only after being authorized.

Table 13-123 ADM_HIST_SQLSTAT columns

Name	Type	Description
INSTANCE_NUMBER	integer	Instance ID of a snapshot.
SQL_ID	bigint	Query ID.
PLAN_HASH_VALUE	bigint	ID of the normalized SQL statement.
MODULE	text	Name of the module that is executing when the SQL statement is first parsed.
ELAPSED_TIME_DELTA	bigint	Valid DB time, which is accumulated if multiple threads are involved (unit: μs)
CPU_TIME_DELTA	bigint	CPU time (unit: μs)
EXECUTIONS_DELTA	integer	Increment in the number of executions that have occurred on this object since it was brought into the cache.
IOWAIT_DELTA	bigint	I/O time (unit: μs).

Name	Type	Description
APWAIT_DELTA	integer	Delta value of the application wait time.
ROWS_PROCESSED_DELTA	bigint	Number of rows in the result set returned by the **SELECT** statement.
SNAP_ID	integer	Unique snapshot ID.

13.3.8 ADM_INDEXES

ADM_INDEXES displays all indexes in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in the **PG_CATALOG** and **SYS** schemas.

Table 13-124 ADM_INDEXES columns

Name	Type	Description
owner	character varying(64)	Index owner
index_name	character varying(64)	Index name
table_name	character varying(64)	Name of a table corresponding to an index
uniqueness	text	Whether an index is unique
partitioned	character(3)	Whether an index has the property of partitioned tables
generated	character varying(1)	Whether the name of an index is generated by the system

13.3.9 ADM_IND_COLUMNS

ADM_IND_COLUMNS displays column information about all indexes in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in the **PG_CATALOG** and **SYS** schemas.

Table 13-125 ADM_IND_COLUMNS columns

Name	Type	Description
index_owner	character varying(64)	Index owner
index_name	character varying(64)	Index name
table_owner	character varying(64)	Table owner
table_name	character varying(64)	Table name
column_name	name	Column name
column_position	smallint	Position of a column in an index

13.3.10 ADM_IND_EXPRESSIONS

ADM_IND_EXPRESSIONS displays information about expression indexes in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in the **PG_CATALOG** and **SYS** schemas.

Table 13-126 ADM_IND_EXPRESSIONS columns

Name	Type	Description
table_owner	character varying(64)	Table owner
table_name	character varying(64)	Table name
index_owner	character varying(64)	Index owner
index_name	character varying(64)	Index name
column_expression	text	Function-based index expression of a specified column
column_position	smallint	Position of a column in an index

13.3.11 ADM_IND_PARTITIONS

ADM_IND_PARTITIONS displays information about all index partitions in the database (excluding global indexes on partitioned tables). Each index partition of a partitioned table in the database, if present, has a row of records in **ADM_IND_PARTITIONS**. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both **PG_CATALOG** and **SYS schema**.

Table 13-127 ADM_IND_PARTITIONS columns

Name	Type	Description
index_owner	character varying(64)	Owner name of the partitioned table index to which an index partition belongs
index_name	character varying(64)	Index name of the partitioned table index to which an index partition belongs
partition_name	character varying(64)	Name of an index partition
def_tablespace_name	name	Tablespace name of an index partition
high_value	text	Limit of the partition corresponding to an index partition **NOTE** ● For range partitioning and interval partitioning, the upper limit of each partition is displayed. ● For list partitioning, the value list of each partition is displayed. ● For hash partitioning, the number of each partition is displayed.
index_partition_usable	boolean	Whether an index partition is available ● **t** (true): yes ● **f** (false): no
schema	character varying(64)	Schema of the partitioned table index to which an index partition belongs
high_value_length	integer	Character length of the limit of the partition corresponding to an index partition.

13.3.12 ADM_IND_SUBPARTITIONS

ADM_IND_SUBPARTITIONS displays information about level-2 partitions of all indexes (excluding global indexes of partitioned tables) in the database. Each level-2 index partition of a level-2 partitioned table in the database, if present, has a row of records in **ADM_IND_SUBPARTITIONS**. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in the **PG_CATALOG** and **SYS** schemas.

Table 13-128 ADM_IND_SUBPARTITIONS columns

Name	Type	Description
index_owner	character varying(64)	Owner name of the partitioned table index to which an index partition belongs
index_name	character varying(64)	Index name of the partitioned table index to which an index partition belongs
partition_name	character varying(64)	Name of the partition where an index is located
subpartition_name	character varying(64)	Name of the level-2 partition where an index is located
def_tablespace_name	name	Tablespace name of an index partition
high_value	text	Limit of the partition corresponding to an index partition **NOTE** For range partitioning and interval partitioning, the upper limit of each partition is displayed. For list partitioning, the value list of each partition is displayed. For hash partitioning, the number of each partition is displayed.
index_partition_usable	boolean	Whether an index partition is available ● **t** (true): yes ● **f** (false): no
schema	character varying(64)	Schema of the partitioned table index to which an index partition belongs
high_value_length	integer	Character length of the limit of the partition corresponding to an index partition

13.3.13 ADM_PART_INDEXES

ADM_PART_INDEXES displays information about all partitioned table indexes (excluding global indexes of partitioned tables) in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both **PG_CATALOG** and **SYS schema**.

Table 13-129 ADM_PART_INDEXES columns

Name	Type	Description
def_tablespace_name	name	Tablespace name of a partitioned table index
index_owner	character varying(64)	Owner name of a partitioned table index
index_name	character varying(64)	Name of a partitioned table index
partition_count	bigint	Number of index partitions of a partitioned table index
partitioning_key_count	integer	Number of partition keys of a partitioned table
partitioning_type	text	Partitioning policy of a partitioned table **NOTE** For details about the supported partitioning policies of the current partitioned table, see **CREATE TABLE PARTITION**.
schema	character varying(64)	Name of the schema to which a partitioned table index belongs
table_name	character varying(64)	Name of the partitioned table to which a partitioned table index belongs
subpartitioning_type	text	Partitioning policy of a level-2 partitioned table. If the partitioned table is a level-1 partitioned table, **NONE** is displayed. **NOTE** For details about the supported partitioning policies of the current level-2 partitioned table, see **CREATE TABLE SUBPARTITION**.

Name	Type	Description
def_subpartition_count	integer	Default number of level-2 partitions to be created. The value is **1** for a level-2 partitioned table and **0** for a level-1 partitioned table.
subpartitioning_key_count	integer	Number of level-2 partition keys of the partitioned table.

13.3.14 ADM_OBJECTS

ADM_OBJECTS displays all database objects in the database. Only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both **PG_CATALOG** and **SYS schema**.

Table 13-130 ADM_OBJECTS columns

Name	Type	Description
owner	name	Object owner
object_name	name	Object name
object_id	oid	Object OID
object_type	name	Object class For example, table, schema, and index.
namespace	oid	Namespace containing an object
created	timestamp with time zone	Creation time of an object
last_ddl_time	timestamp with time zone	Last modification time of an object

NOTICE

For details about the value ranges of **created** and **last_ddl_time**, see PG_OBJECT.

13.3.15 ADM_PART_TABLES

ADM_PART_TABLES displays information about all partitioned tables in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both **PG_CATALOG** and **SYS schema**.

Table 13-131 ADM_PART_TABLES columns

Name	Type	Description
table_owner	character varying(64)	Owner name of a partitioned table
table_name	character varying(64)	Name of a partitioned table
partitioning_type	text	Partitioning policy of a partitioned table **NOTE** For details about the supported partitioning policies of the current partitioned table, see **CREATE TABLE PARTITION**.
partition_count	bigint	Number of partitions of a partitioned table
partitioning_key_count	integer	Number of partition keys of a partitioned table
def_tablespace_name	name	Tablespace name of a partitioned table
schema	character varying(64)	Schema of a partitioned table
subpartitioning_type	text	Partitioning policy of a level-2 partitioned table. If the partitioned table is a level-1 partitioned table, **NONE** is displayed. **NOTE** For details about the supported partitioning policies of the current level-2 partitioned table, see **CREATE TABLE SUBPARTITION**.
def_subpartition_count	integer	Default number of level-2 partitions to be created. The value is **1** for a level-2 partitioned table and **0** for a level-1 partitioned table.
subpartitioning_key_count	integer	Number of level-2 partition keys of the partitioned table.

13.3.16 ADM_PROCEDURES

ADM_PROCEDURES displays information about all stored procedures and functions in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in the **PG_CATALOG** and **SYS** schemas.

Table 13-132 ADM_PROCEDURES columns

Name	Type	Description
owner	character varying(64)	Owner of a stored procedure or function
object_name	character varying(64)	Name of a stored procedure or function
argument_number	smallint	Number of input parameters in a stored procedure

13.3.17 ADM_SCHEDULER_JOBS

ADM_SCHEDULER_JOBS displays information about all DBE_SCHEDULER scheduled tasks in the database.

Table 13-133 ADM_SCHEDULER_JOBS columns

Name	Type	Description
owner	name	Owner of a scheduled task.
job_name	text	Name of a scheduled task.
job_style	text	Action mode of a scheduled task.
job_creator	name	Creator of a scheduled task.
program_name	text	Name of the program referenced by a scheduled task.
job_action	text	Program content of a scheduled task.
number_of_arguments	text	Number of parameters of a scheduled task.
schedule_name	text	Name of the schedule referenced by a scheduled task.
start_date	timestamp without time zone	Start time of a scheduled task.
repeat_interval	text	Period of a scheduled task.

Name	Type	Description
end_date	timestamp without time zone	End time of a scheduled task.
job_class	text	Name of the scheduled task class to which a scheduled task belongs.
enabled	boolean	Status of a scheduled task.
auto_drop	text	Status of the automatic deletion function of a scheduled task.
state	"char"	Status of a scheduled task.
failure_count	smallint	Number of scheduled task failures.
last_start_date	timestamp without time zone	Last time when a scheduled task is started.
next_run_date	timestamp without time zone	Next execution time of a scheduled task.
destination	text	Target name of a scheduled task.
credential_name	text	Certificate name of a scheduled task.
comments	text	Comments of a scheduled task.

13.3.18 ADM_SEQUENCES

ADM_SEQUENCES displays information about all sequences in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both **PG_CATALOG** and **SYS schema**.

Table 13-134 ADM_SEQUENCES columns

Name	Type	Description
sequence_owner	character varying(64)	Owner of a sequence
sequence_name	character varying(64)	Name of a sequence
min_value	int16	Minimum value of a sequence

Name	Type	Description
max_value	int16	Maximum value of a sequence
increment_by	int16	Increment of a sequence
last_number	int16	Value of the previous sequence
cache_size	int16	Size of the sequence disk cache
cycle_flag	character(1)	Whether the sequence is a cycle sequence. The value can be **Y** or **N**. ● **Y**: It is a cycle sequence. ● **N**: It is not a cycle sequence.

13.3.19 ADM_SOURCE

ADM_SOURCE displays all stored procedures or functions in the database and provides columns defined by the stored procedures or functions. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in the **PG_CATALOG** and **SYS** schemas.

Table 13-135 ADM_SOURCE columns

Name	Type	Description
owner	character varying(64)	Owner of a stored procedure or function
name	character varying(64)	Name of a stored procedure or function
text	text	Definition of a stored procedure or function

13.3.20 ADM_SYNONYMS

ADM_SYNONYMS displays all synonyms in the database. This view is accessible only to system administrators. This view exists in both **PG_CATALOG** and **SYS schema**.

Table 13-136 ADM_SYNONYMS columns

Name	Type	Description
owner	text	Owner of a synonym
schema_name	text	Name of the schema to which a synonym belongs
synonym_name	text	Name of a synonym
table_owner	text	Owner of an associated object Although the column is called **table_owner**, the associated object owned by the column is not necessarily a table. It can be any common database object, such as a view, stored procedure, or synonym.
table_name	text	Name of an associated object Although the column is called **table_name**, the associated object is not necessarily a table. It can be any common database object, such as a view, stored procedure, or synonym.
table_schema_name	text	Schema name of an associated object Although the column is called **table_schema_name**, the associated object is not necessarily a table. It can be any common database object, such as a view, stored procedure, or synonym.

13.3.21 ADM_TAB_SUBPARTITIONS

ADM_TAB_SUBPARTITIONS displays information about all level-2 partitions in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in the **PG_CATALOG** and **SYS** schemas.

Table 13-137 ADM_TAB_SUBPARTITIONS columns

Name	Type	Description
table_owner	character varying(64)	Role name
table_name	character varying(64)	Relational table name
partition_name	character varying(64)	Partition name
subpartition_name	character varying(64)	Name of a level-2 partition

Name	Type	Description
high_value	text	Limit of a level-2 partition **NOTE** ● For range partitioning and interval partitioning, the upper limit of each partition is displayed. ● For list partitioning, the value list of each partition is displayed. ● For hash partitioning, the number of each partition is displayed.
tablespace_name	name	Tablespace name of a level-2 partitioned table
schema	character varying(64)	Name of a namespace
high_value_length	integer	Character length of the limit of a level-2 partition

13.3.22 ADM_TABLES

ADM_TABLES displays all tables in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in the **PG_CATALOG** and **SYS** schemas.

Table 13-138 ADM_TABLES columns

Name	Type	Description
owner	character varying(64)	Table owner
table_name	character varying(64)	Table name
tablespace_name	character varying(64)	Tablespace name of a table
dropped	character varying	Whether the current record is deleted ● **YES**: It is deleted. ● **NO**: It is not deleted.
num_rows	numeric	Estimated number of rows in a table
status	character varying(8)	Whether the current record is valid

Name	Type	Description
temporary	character(1)	Whether a table is a temporary table ● **Y**: It is a temporary table. ● **N**: It is not a temporary table.

13.3.23 ADM_TABLESPACES

ADM_TABLESPACES displays information about available tablespaces. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in the **PG_CATALOG** and **SYS** schemas.

Table 13-139 ADM_TABLESPACES columns

Name	Type	Description
tablespace_name	character varying(64)	Tablespace name

13.3.24 ADM_TAB_COLUMNS

ADM_TAB_COLUMNS displays columns in tables. Each column in a table of the database has a row in **ADM_TAB_COLUMNS**. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in the **PG_CATALOG** and **SYS** schemas.

Table 13-140 ADM_TAB_COLUMNS columns

Name	Type	Description
owner	character varying(64)	Table owner
table_name	character varying(64)	Table name
column_name	character varying(64)	Column name
data_type	character varying(128)	Data type of a column
data_length	integer	Length of a column, in bytes

Name	Type	Description
data_precision	integer	Precision of a data type. This parameter is valid for the numeric data type and **NULL** for other data types.
data_scale	integer	Number of decimal places. This parameter is valid for the numeric data type and is set to **0** for other data types.
nullable	bpchar	Whether a column can be empty (**n** for the primary key constraint and non-null constraint)
column_id	integer	Sequence number of a column when a table is created
avg_col_len	numeric	Average length of a column, in bytes
char_length	numeric	Column length (in the unit of bytes) which is valid only for varchar, nvarchar2, bpchar, and char types
comments	text	Comments

13.3.25 ADM_TAB_COMMENTS

ADM_TAB_COMMENTS displays comments about all tables and views in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in the **PG_CATALOG** and **SYS** schemas.

Table 13-141 ADM_TAB_COMMENTS columns

Name	Type	Description
owner	character varying(64)	Owner of a table or view
table_name	character varying(64)	Name of a table or view
comments	text	Comments

13.3.26 ADM_TAB_PARTITIONS

ADM_TAB_PARTITIONS stores information about all partitions in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both **PG_CATALOG** and **SYS schema**.

Table 13-142 ADM_TAB_PARTITIONS columns

Name	Type	Description
table_owner	character varying(64)	Role name
table_name	character varying(64)	Relational table name
partition_name	character varying(64)	Name of a partition
high_value	text	Limit of a partition. **NOTE** ● For range partitioning and interval partitioning, the upper limit of each partition is displayed. ● For list partitioning, the value list of each partition is displayed. ● For hash partitioning, the number of each partition is displayed.
tablespace_name	name	Tablespace name of a partitioned table
schema	character varying(64)	Name of a namespace
subpartition_count	bigint	Number of level-2 partitions.
high_value_length	integer	Character length of the limit of a partition.

13.3.27 ADM_TRIGGERS

ADM_TRIGGERS displays information about triggers in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in the **PG_CATALOG** and **SYS** schemas.

Table 13-143 ADM_TRIGGERS columns

Name	Type	Description
trigger_name	character varying(64)	Trigger name
table_owner	character varying(64)	Role name
table_name	character varying(64)	Relational table name

Name	Type	Description
status	character varying(64)	● **O**: The trigger is enabled in origin or local mode. ● **D**: The trigger is disabled. ● **R**: The trigger is enabled in replica mode. ● **A**: The trigger is always enabled.

13.3.28 ADM_TYPE_ATTRS

ADM_TYPE_ATTRS displays the attributes of the current database object type. By default, only the system administrator can access this view. Common users can access the view only after being authorized.

Table 13-144 ADM_TYPE_ATTRS columns

Name	Type	Description
OWNER	oid	Owner of the type
TYPE_NAME	name	Data type name
ATTR_NAME	name	Column name
ATTR_TYPE_MOD	integer	Type-specific data supplied at the table creation time (for example, the maximum length of a **varchar** column). This column is used as the third parameter when passing to type-specific input functions and length coercion functions. The value will generally be **–1** for types that do not need **atttypmod**.
ATTR_TYPE_OWNER	oid	Owner of an attribute of this type.
ATTR_TYPE_NAME	name	Data type attribute name.
LENGTH	smallint	Number of bytes in the internal representation of the type for a fixed-size type. It is a negative number for a variable-length type. ● The value **–1** indicates a "varlena" type (one that has a length word). ● The value **–2** indicates a null-terminated C string.
PRECISION	integer	Precision of the numeric type.
SCALE	integer	Range of the numeric type.

Name	Type	Description
CHARACTER_SET_NAME	character(1)	Character set name of an attribute (**c** or **n**).
ATTR_NO	smalllint	Attribute number
INHERITED	character(1)	Determines whether the attribute is inherited from the super type (**Y** or **N**).

13.3.29 ADM_USERS

ADM_USERS displays all usernames in the database. This view is accessible only to system administrators. This view exists in both **PG_CATALOG** and **SYS schema**.

Table 13-145 ADM_USERS columns

Name	Type	Description
username	character varying(64)	Username

13.3.30 ADM_VIEWS

ADM_VIEWS displays views in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in the **PG_CATALOG** and **SYS** schemas.

Table 13-146 ADM_VIEWS columns

Name	Type	Description
owner	character varying(64)	View owner
view_name	character varying(64)	View name

13.3.31 DB_ALL_TABLES

DB_ALL_TABLES displays tables or views accessible to the current user. This view exists in both **PG_CATALOG** and **SYS schema**.

Table 13-147 DB_ALL_TABLES columns

Name	Type	Description
owner	name	Owner of a table or view
table_name	name	Name of a table or view

Name	Type	Description
tablespace_name	name	Tablespace where a table or view is located

13.3.32 DB_COL_COMMENTS

DB_COL_COMMENTS displays comment information about table columns accessible to the current user. This view exists in both **PG_CATALOG** and **SYS schema**.

Table 13-148 DB_COL_COMMENTS columns

Name	Type	Description
owner	character varying(64)	Table owner
table_name	character varying(64)	Table name
column_name	character varying(64)	Column name
comments	text	Comments

13.3.33 DB_CONSTRAINTS

DB_CONSTRAINTS displays information about constraints accessible to the current user. This view exists in both **PG_CATALOG** and **SYS schema**.

Table 13-149 DB_CONSTRAINTS columns

Name	Type	Description
owner	character varyiny(64)	Constraint creator
constraint_name	character varying(64)	Constraint name
constraint_type	text	Constraint type ● **c**: check constraint ● **f**: foreign key constraint ● **p**: primary key constraint ● **u**: unique constraint
table_name	character varying(64)	Name of a constraint-related table
index_owner	character varying(64)	Owner of a constraint-related index (only for the unique constraint and primary key constraint)

Name	Type	Description
index_name	character varying(64)	Name of a constraint-related index (only for the unique constraint and primary key constraint)

13.3.34 DB_CONS_COLUMNS

DB_CONS_COLUMNS displays information about constraint columns accessible to the current user. This view exists in both **PG_CATALOG** and **SYS schema**.

Table 13-150 DB_CONS_COLUMNS columns

Name	Type	Description
owner	character varying(64)	Constraint creator
constraint_name	character varying(64)	Constraint name
table_name	character varying(64)	Name of a constraint-related table
column_name	character varying(64)	Name of a constraint-related column
position	smallint	Position of a column in a table

13.3.35 DB_DEPENDENCIES

DB_DEPENDENCIES displays dependencies between functions and advanced packages accessible to the current user. This view exists in both **PG_CATALOG** and **SYS schema**.

NOTICE

In GaussDB, this table is empty without any record due to information constraints.

Table 13-151 DB_DEPENDENCIES columns

Name	Type	Description
owner	character varying(30)	Object owner
name	character varying(30)	Object name
type	character varying(17)	Object class

Name	Type	Description
referenced_owner	character varying(30)	Owner of a referenced object
referenced_name	character varying(64)	Name of a referenced object
referenced_type	character varying(17)	Type of a referenced object
referenced_link_name	character varying(128)	Name of the link to a referenced object
schemaid	numeric	ID of the current schema
dependency_type	character varying(4)	Dependency type (**REF** indicates soft reference and **HARD** indicates direct description.)

13.3.36 DB_IND_COLUMNS

DB_IND_COLUMNS displays all index columns accessible to the current user. This view exists in both **PG_CATALOG** and **SYS schema**.

Table 13-152 DB_IND_COLUMNS columns

Name	Type	Description
index_owner	character varying(64)	Index owner
index_name	character varying(64)	Index name
table_owner	character varying(64)	Table owner
table_name	character varying(64)	Table name
column_name	name	Column name
column_position	smallint	Position of a column in an index

13.3.37 DB_IND_EXPRESSIONS

DB_IND_EXPRESSIONS displays information about expression indexes accessible to the current user. This view exists in both **PG_CATALOG** and **SYS schema**.

Table 13-153 DB_IND_EXPRESSIONS columns

Name	Type	Description
index_owner	character varying(64)	Index owner

Name	Type	Description
index_name	character varying(64)	Index name
table_owner	character varying(64)	Table owner
table_name	character varying(64)	Table name
column_expression	text	Function-based index expression of a specified column
column_position	smallint	Position of a column in an index

13.3.38 DB_IND_PARTITIONS

DB_IND_PARTITIONS displays information about partitioned table indexes (excluding global indexes of partitioned tables) accessible to the current user. This view exists in the **PG_CATALOG** and **SYS** schemas.

Table 13-154 DB_IND_PARTITIONS columns

Name	Type	Description
index_owner	character varying(64)	Owner name of the partitioned table index to which an index partition belongs
index_name	character varying(64)	Index name of the partitioned table index to which an index partition belongs
partition_name	character varying(64)	Name of the partition where an index is located
def_tablespace_name	name	Tablespace name of an index partition

Name	Type	Description
high_value	text	Limit of the partition corresponding to an index partition **NOTE** ● For range partitioning and interval partitioning, the upper limit of each partition is displayed. ● For list partitioning, the value list of each partition is displayed. ● For hash partitioning, the number of each partition is displayed.
index_partition_usable	boolean	Whether an index partition is available ● **t** (true): yes ● **f** (false): no
schema	character varying(64)	Schema of the partitioned table index to which an index partition belongs
high_value_length	integer	Character length of the limit of the partition corresponding to an index partition

13.3.39 DB_IND_SUBPARTITIONS

DB_IND_SUBPARTITIONS displays information about level-2 partitions of indexes (excluding global indexes of partitioned tables) accessible to the current user. This view exists in the **PG_CATALOG** and **SYS** schemas.

Table 13-155 DB_IND_SUBPARTITIONS columns

Name	Type	Description
index_owner	character varying(64)	Owner name of the partitioned table index to which an index partition belongs
index_name	character varying(64)	Index name of the partitioned table index to which an index partition belongs
partition_name	character varying(64)	Name of an index partition

Name	Type	Description
subpartition_name	character varying(64)	Name of a level-2 index partition
def_tablespace_name	name	Tablespace name of an index partition
high_value	text	Limit of the partition corresponding to an index partition **NOTE** ● For range partitioning and interval partitioning, the upper limit of each partition is displayed. ● For list partitioning, the value list of each partition is displayed. ● For hash partitioning, the number of each partition is displayed.
index_partition_usable	boolean	Whether an index partition is available ● **t** (true): yes ● **f** (false): no
schema	character varying(64)	Schema of the partitioned table index to which an index partition belongs
high_value_length	integer	Character length of the limit of the partition corresponding to an index partition

13.3.40 DB_INDEXES

DB_INDEXES displays information about indexes accessible to the current user. This view exists in both **PG_CATALOG** and **SYS schema**.

Table 13-156 DB_INDEXES columns

Name	Type	Description
owner	character varying(64)	Index owner
index_name	character varying(64)	Index name
table_name	character varying(64)	Name of a table corresponding to an index

Name	Type	Description
uniqueness	text	Whether an index is unique
partitioned	character(3)	Whether an index has the property of partitioned tables
generated	character varying(1)	Whether the name of an index is generated by the system

13.3.41 DB_OBJECTS

DB_OBJECTS displays all database objects accessible to the current user. This view exists in both **PG_CATALOG** and **SYS schema**.

Table 13-157 DB_OBJECTS columns

Name	Type	Description
owner	name	Object owner
object_name	name	Object name
object_id	oid	Object OID
object_type	name	Object class
namespace	oid	ID of the namespace where an object resides
created	timestamp with time zone	Creation time of an object
last_ddl_time	timestamp with time zone	Last modification time of an object

NOTICE

For details about the value ranges of **created** and **last_ddl_time**, see PG_OBJECT.

13.3.42 DB_PROCEDURES

DB_PROCEDURES displays information about all stored procedures or functions accessible to the current user. This view exists in both **PG_CATALOG** and **SYS schema**.

Table 13-158 DB_PROCEDURES columns

Name	Type	Description
owner	name	Object owner
object_name	name	Object name

13.3.43 DB_PART_INDEXES

DB_PART_INDEXES displays information about partitioned table indexes (excluding global indexes of partitioned tables) accessible to the current user. This view exists in the **PG_CATALOG** and **SYS** schemas.

Table 13-159 DB_PART_INDEXES columns

Name	Type	Description
def_tablespace_name	name	Tablespace name of a partitioned table index
index_owner	character varying(64)	Owner name of a partitioned table index
index_name	character varying(64)	Name of a partitioned table index
partition_count	bigint	Number of index partitions of a partitioned table index
partitioning_key_count	integer	Number of partition keys of a partitioned table
partitioning_type	text	Partitioning policy of a partitioned table **NOTE** For details about the supported partitioning policies of the current partitioned table, see **CREATE TABLE PARTITION**.
schema	character varying(64)	Name of the schema to which a partitioned table index belongs
table_name	character varying(64)	Name of the partitioned table to which a partitioned table index belongs

Name	Type	Description
subpartitioning_type	text	Partitioning policy of a level-2 partitioned table. If the partitioned table is a level-1 partitioned table, **NONE** is displayed. **NOTE** For details about the supported partitioning policies of the current level-2 partitioned table, see **CREATE TABLE SUBPARTITION**.
def_subpartition_count	integer	Default number of level-2 partitions to be created. The value is **1** for a level-2 partitioned table and **0** for a level-1 partitioned table.
subpartitioning_key_count	integer	Number of level-2 partition keys of a partitioned table

13.3.44 DB_PART_TABLES

DB_PART_TABLES displays information about partitioned tables accessible to the current user. This view exists in the **PG_CATALOG** and **SYS** schemas.

Table 13-160 DB_PART_TABLES columns

Name	Type	Description
table_owner	character varying(64)	Owner name of a partitioned table
table_name	character varying(64)	Name of a partitioned table
partitioning_type	text	Partitioning policy of a partitioned table **NOTE** For details about the supported partitioning policies of the current partitioned table, see **CREATE TABLE PARTITION**.
partition_count	bigint	Number of partitions of a partitioned table
partitioning_key_count	integer	Number of partition keys of a partitioned table
def_tablespace_name	name	Tablespace name of a partitioned table

Name	Type	Description
schema	character varying(64)	Schema of a partitioned table
subpartitioning_type	text	Partitioning policy of a level-2 partitioned table. If the partitioned table is a level-1 partitioned table, **NONE** is displayed. **NOTE** For details about the supported partitioning policies of the current level-2 partitioned table, see **CREATE TABLE SUBPARTITION**.
def_subpartition_count	integer	Default number of level-2 partitions to be created. The value is **1** for a level-2 partitioned table and **0** for a level-1 partitioned table.
subpartitioning_key_count	integer	Number of level-2 partition keys of a partitioned table

13.3.45 DB_SEQUENCES

DB_SEQUENCES displays all sequences accessible to the current user. This view exists in the **PG_CATALOG** and **SYS** schemas.

Table 13-161 DB_SEQUENCES columns

Name	Type	Description
sequence_owner	name	Owner of a sequence
sequence_name	name	Name of a sequence
min_value	int16	Minimum value of a sequence
max_value	int16	Maximum value of a sequence
increment_by	int16	Value by which a sequence is incremented
cycle_flag	character(1)	Whether the sequence is a cycle sequence. The value can be **Y** or **N**. ● **Y**: It is a cycle sequence. ● **N**: It is not a cycle sequence.
last_number	int16	Value of the previous sequence

Name	Type	Description
cache_size	int16	Size of the sequence disk cache

13.3.46 DB_SOURCE

DB_SOURCE displays information about stored procedures or functions accessible to the current user, and provides columns defined by the stored procedures and functions. This view exists in both **PG_CATALOG** and **SYS schema**.

Table 13-162 DB_SOURCE columns

Name	Type	Description
owner	name	Object owner
name	name	Object name
type	name	Object class
text	text	Object definition

13.3.47 DB_SYNONYMS

DB_SYNONYMS displays all synonyms accessible to the current user.

Table 13-163 DB_SYNONYMS columns

Name	Type	Description
owner	text	Owner of a synonym
schema_name	text	Name of the schema to which a synonym belongs
synonym_name	text	Name of a synonym
table_owner	text	Owner of an associated object Although the column is called **table_owner**, the associated object owned by the column is not necessarily a table. It can be any common database object, such as a view, stored procedure, or synonym.
table_name	text	Name of an associated object Although the column is called **table_name**, the associated object is not necessarily a table. It can be any common database object, such as a view, stored procedure, or synonym.

Name	Type	Description
table_schema_name	text	Schema name of an associated object Although the column is called **table_schema_name**, the associated object is not necessarily a table. It can be any common database object, such as a view, stored procedure, or synonym.

13.3.48 DB_TAB_PARTITIONS

DB_TAB_PARTITIONS displays information about partitioned tables accessible to the current user. This view exists in the **PG_CATALOG** and **SYS** schemas.

Table 13-164 DB_TAB_PARTITIONS columns

Name	Type	Description
table_owner	character varying(64)	Role name
table_name	character varying(64)	Relational table name
partition_name	character varying(64)	Partition name
high_value	text	Limit of a partition **NOTE** ● For range partitioning and interval partitioning, the upper limit of each partition is displayed. ● For list partitioning, the value list of each partition is displayed. ● For hash partitioning, the number of each partition is displayed.
tablespace_name	name	Tablespace name of a partitioned table
schema	character varying(64)	Name of a namespace
subpartition_count	bigint	Number of level-2 partitions
high_value_length	integer	Character length of the limit of a partition

13.3.49 DB_TAB_SUBPARTITIONS

DB_TAB_SUBPARTITIONS displays information about level-2 partitioned tables accessible to the current user. This view exists in the **PG_CATALOG** and **SYS** schemas.

Table 13-165 DB_TAB_SUBPARTITIONS columns

Name	Type	Description
table_owner	character varying(64)	Role name
table_name	character varying(64)	Relational table name
partition_name	character varying(64)	Partition name
subpartition_name	character varying(64)	Name of a level-2 partition
high_value	text	Limit of a level-2 partition **NOTE** ● For range partitioning and interval partitioning, the upper limit of each partition is displayed. ● For list partitioning, the value list of each partition is displayed. ● For hash partitioning, the number of each partition is displayed.
tablespace_name	name	Tablespace name of a level-2 partitioned table
schema	character varying(64)	Name of a namespace
high_value_length	integer	Character length of the limit of a level-2 partition

13.3.50 DB_TAB_COLUMNS

DB_TAB_COLUMNS displays description information about columns of tables accessible to the current user. This view exists in both **PG_CATALOG** and **SYS schema**.

Table 13-166 DB_TAB_COLUMNS columns

Name	Type	Description
owner	character varying(64)	Table owner
table_name	character varying(64)	Table name
column_name	character varying(64)	Column name
data_type	character varying(128)	Data type of a column

Name	Type	Description
data_length	integer	Length of a column, in bytes
data_precision	integer	Precision of the data type. This parameter is valid for the numeric data type and is set to **NULL** for other data types.
data_scale	integer	Number of decimal places. This parameter is valid for the numeric data type and is set to **0** for other data types.
nullable	bpchar	Whether a column can be empty (**n** for the primary key constraint and non-null constraint)
column_id	integer	Column ID generated when an object is created or a column is added
char_length	numeric	Length of a column, in characters. This parameter is valid only for the varchar, nvarchar2, bpchar, and char types.
avg_col_len	numeric	Average length of a column, in bytes
comments	text	Comments

13.3.51 DB_TAB_COMMENTS

DB_TAB_COMMENTS displays comments about all tables and views accessible to the current user. This view exists in both **PG_CATALOG** and **SYS schema**.

Table 13-167 DB_TAB_COMMENTS columns

Name	Type	Description
owner	character varying(64)	Owner of a table or view
table_name	character varying(64)	Name of a table or view
comments	text	Comments

13.3.52 DB_TABLES

DB_TABLES displays all tables accessible to the current user. This view exists in both **PG_CATALOG** and **SYS schema**.

Table 13-168 DB_TABLES columns

Name	Type	Description
owner	character varying(64)	Table owner
table_name	character varying(64)	Table name
tablespace_name	character varying(64)	Tablespace name of a table
num_rows	numeric	Estimated number of rows in a table
status	character varying(8)	Whether the current record is valid
temporary	character(1)	Whether a table is a temporary table ● **Y**: It is a temporary table. ● **N**: It is not a temporary table.
dropped	character varying	Whether the current record is deleted ● **YES**: It is deleted. ● **NO**: It is not deleted.

13.3.53 DB_TRIGGERS

DB_TRIGGERS displays information about triggers accessible to the current user. This view exists in the **PG_CATALOG** and **SYS** schemas.

Table 13-169 DB_TRIGGERS columns

Name	Type	Description
trigger_name	character varying(64)	Trigger name
table_owner	character varying(64)	Role name
table_name	character varying(64)	Relational table name
status	character varying(64)	● **O**: The trigger is enabled in origin or local mode. ● **D**: The trigger is disabled. ● **R**: The trigger is enabled in replica mode. ● **A**: The trigger is always enabled.

13.3.54 DB_USERS

DB_USERS displays all users of the database visible to the current user. However, it does not describe the users. By default, only the system administrator can access this view. This view exists in both **PG_CATALOG** and **SYS schema**.

Table 13-170 DB_USERS columns

Name	Type	Description
user_id	oid	OID of a user
username	name	Username

13.3.55 DB_VIEWS

DB_VIEWS displays the description about all views accessible to the current user. This view exists in both **PG_CATALOG** and **SYS schema**.

Table 13-171 DB_VIEWS columns

Name	Type	Description
owner	name	View owner
view_name	name	View name
text	text	View text
text_length	integer	Text length of a view

13.3.56 DV_SESSIONS

DV_SESSIONS displays all session information about the current session. By default, only the system administrator can access this view. Common users can access the view only after being authorized.

Table 13-172 DV_SESSIONS columns

Name	Type	Description
sid	bigint	OID of the active backend thread of the current session
serial#	integer	Sequence number of the backend thread of the current activity, which is **0** in GaussDB
user#	oid	OID of the user that has logged in to the backend thread. The OID is **0** if the backend thread is a global auxiliary thread.

Name	Type	Description
username	name	Name of the user logged in to the backend process. The value is null if the backend thread is a global auxiliary thread. **application_name** can be identified by associating with **pg_stat_get_activity()**. Example: select s.*,a.application_name from DV_SESSIONS as s left join pg_stat_get_activity(NULL) as a on s.sid=a.sessionid;

13.3.57 DV_SESSION_LONGOPS

DV_SESSION_LONGOPS displays the progress of ongoing operations. Users can access this view only after being authorized.

Table 13-173 DV_SESSION_LONGOPS columns

Name	Type	Description
sid	bigint	OID of the running backend process
serial#	integer	Sequence number of the running backend process, which is **0** in GaussDB
sofar	integer	Completed workload, which is null in GaussDB
totalwork	integer	Total workload, which is null in GaussDB

13.3.58 GET_GLOBAL_PREPARED_XACTS (Discarded)

This view is not supported in the centralized system.

13.3.59 GS_ALL_CONTROL_GROUP_INFO

This view is not supported in the centralized system.

13.3.60 GS_AUDITING

GS_AUDITING displays all audit information about database-related operations. Only the users with system administrator or security policy administrator permission can access this view.

Table 13-174 GS_AUDITING columns

Name	Type	Description
polname	name	Policy name, which must be unique
pol_type	text	Audit policy type. The value can be **access** or **privilege**. ● **access**: DML operations are audited. ● **privilege**: DDL operations are audited.
polenabled	boolean	Specifies whether to enable a policy. ● **t** (true): enabled ● **f** (false): disabled
access_type	name	DML database operation type. For example, SELECT, INSERT, and DELETE.
label_name	name	Resource label name. This column corresponds to the **polname** column in the **GS_AUDITING_POLICY** system catalog.
priv_object	text	Describes the path of the database asset.
filter_name	text	Logical character string of a filter criterion

13.3.61 GS_AUDITING_ACCESS

GS_AUDITING_ACCESS displays all audit information about database DML-related operations. Only the users with system administrator or security policy administrator permission can access this view.

Table 13-175 GS_AUDITING_ACCESS columns

Name	Type	Description
polname	name	Policy name, which must be unique
pol_type	text	Audit policy type. The value **access** indicates that DML operations are audited.
polenabled	boolean	Specifies whether to enable a policy.
access_type	name	DML database operation type. For example, SELECT, INSERT, and DELETE.
label_name	name	Resource label name. This column corresponds to the **polname** column in the **GS_AUDITING_POLICY** system catalog.
access_object	text	Describes the path of the database asset.
filter_name	text	Logical character string of a filter criterion

13.3.62 GS_AUDITING_PRIVILEGE

GS_AUDITING_PRIVILEGE displays all audit information about database DDL-related operations. Only the users with system administrator or security policy administrator permission can access this view.

Table 13-176 GS_AUDITING_PRIVILEGE columns

Name	Type	Description
polname	name	Policy name, which must be unique
pol_type	text	Audit policy type. The value **privilege** indicates that DDL operations are audited.
polenabled	boolean	Specifies whether to enable a policy.
access_type	name	DDL database operation type. For example, CREATE, ALTER, and DROP.
label_name	name	Resource label name. This column corresponds to the **polname** column in the **GS_AUDITING_POLICY** system catalog.
priv_object	text	Full domain name of a database object
filter_name	text	Logical character string of a filter criterion

13.3.63 GS_CLUSTER_RESOURCE_INFO

This view is not supported in the centralized system.

13.3.64 GS_COMM_PROXY_THREAD_STATUS

GS_COMM_PROXY_THREAD_STATUS displays statistics on packets sent and received by the proxy communication library **comm_proxy**. This view displays statistics about data sent and received by **comm_proxy** only when the user-mode network deployment mode is enabled during the installation of the centralized database and **enable_dfx** of **comm_proxy_attr** is set to **true**. In other scenarios, this view cannot be queried.

Table 13-177 GS_COMM_PROXY_THREAD_STATUS columns

Name	Type	Description
ProxyThreadId	bigint	ID of the current network proxy thread **comm_proxy**.
ProxyCpuAffinity	text	NUMA-CPU affinity of the current network proxy thread **comm_proxy**, indicating the NUMA and CPU ID.
ThreadStartTime	text	Start time of the current network proxy thread **comm_proxy**.

Name	Type	Description
RxPckNums	bigint	Number of packets received by the current network proxy thread **comm_proxy**.
TxPckNums	bigint	Number of packets sent by the current network proxy thread **comm_proxy**.
RxPcks	float	Number of packets received by the network proxy thread **comm_proxy** per second.
TxPcks	float	Number of packets sent by the network proxy thread **comm_proxy** per second.

13.3.65 GS_DB_PRIVILEGES

GS_DB_PRIVILEGES displays the granting of ANY permissions. Each record corresponds to a piece of authorization information.

Table 13-178 GS_DB_PRIVILEGES columns

Name	Type	Description
rolename	name	Username.
privilege_type	text	ANY permission of a user. For details about the value, see **Table 11-122**.
admin_option	text	Whether the ANY permission recorded in the **privilege_type** column can be re-granted. ● **yes** ● **no**

13.3.66 GS_FILE_STAT

GS_FILE_STAT records statistics about data file I/O to indicate I/O performance and detect performance problems such as abnormal I/O operations.

Table 13-179 GS_FILE_STAT columns

Name	Type	Description
filenum	oid	File identifier
dbid	oid	Database ID
spcid	oid	Tablespace ID
phyrds	bigint	Number of times of reading physical files

Name	Type	Description
phywrts	bigint	Number of times of writing into physical files
phyblkrd	bigint	Number of times of reading physical file blocks
phyblkwrt	bigint	Number of times of writing into physical file blocks
readtim	bigint	Total duration of reading, in microseconds
writetim	bigint	Total duration of writing, in microseconds
avgiotim	bigint	Average duration of reading and writing, in microseconds
lstiotim	bigint	Duration of the last file reading, in microseconds
miniotim	bigint	Minimum duration of reading and writing, in microseconds
maxiowtm	bigint	Maximum duration of reading and writing, in microseconds

13.3.67 GS_GET_CONTROL_GROUP_INFO

This view is not supported in the centralized system.

13.3.68 GS_GSC_MEMORY_DETAIL

GS_GSC_MEMORY_DETAIL displays the memory usage of the global system cache of the current process on the current node. This view is displayed only when GSC is enabled. Note that the query is separated by the database memory context. Therefore, some memory statistics are missing. The memory context corresponding to the missing memory statistics is **GlobalSysDBCache**.

The current feature is a lab feature. Contact Huawei technical support before using it.

Table 13-180 GS_GSC_MEMORY_DETAIL columns

Name	Type	Description
db_id	text	Database ID.
totalsize	numeric	Total size of the shared memory, in bytes.
freesize	numeric	Remaining size of the shared memory, in bytes.
usedsize	numeric	Used size of the shared memory, in bytes.

13.3.69 GS_INSTANCE_TIME

GS_INSTANCE_TIME records time consumption information of the current node. The time consumption information is classified into the following types:

- DB_TIME: effective time spent by jobs in multi-core scenarios.
- CPU_TIME: CPU time cost.
- EXECUTION_TIME: time spent in the executor.
- PARSE_TIME: time spent on parsing SQL statements.
- PLAN_TIME: time spent on generating plans.
- REWRITE_TIME: time spent on rewriting SQL statements.
- PL_EXECUTION_TIME: execution time of the PL/pgSQL stored procedure.
- PL_COMPILATION_TIME: compilation time of the PL/pgSQL stored procedure.
- NET_SEND_TIME: time spent on the network.
- DATA_IO_TIME: time spent on I/Os.

Table 13-181 GS_INSTANCE_TIME columns

Name	Type	Description
stat_id	integer	Statistics ID
stat_name	text	Type name
value	bigint	Time value, in μs

13.3.70 GS_LABELS

GS_LABELS displays all configured resource labels. Only the users with system administrator or security policy administrator permission can access this view.

Name	Type	Description
labelname	name	Resource label name
labeltype	name	Resource label type This parameter corresponds to the **labeltype** column in the **GS_POLICY_LABEL** system catalog.
fqdntype	name	Database resource type Such as table, schema, and index.
schemaname	name	Name of the schema to which the database resource belongs
fqdnname	name	Database resource name
columnname	name	Name of the database resource column. If the marked database resource is not a column, this parameter is left blank.

13.3.71 GS_LSC_MEMORY_DETAIL

GS_LSC_MEMORY_DETAIL collects statistics about the local SysCache memory usage of all threads based on the MemoryContext node. The statistics are available only when GSC is enabled.

The current feature is a lab feature. Contact Huawei technical support before using it.

Table 13-182 GS_LSC_MEMORY_DETAIL columns

Name	Type	Description
threadid	text	Thread start time + thread ID (string: *timestamp.sessionid*)
tid	bigint	Thread ID
thrdtype	text	Thread type. It can be any thread type in the system, such as postgresql and wlmmonitor.
contextname	text	Name of the memory context
level	smallint	Hierarchy of the memory context
parent	text	Name of the parent memory context
totalsize	bigint	Total size of the memory context, in bytes
freesize	bigint	Total size of released memory in the current memory context, in bytes
usedsize	bigint	Total size of used memory in the current memory context, in bytes

13.3.72 GS_MASKING

GS_MASKING displays all configured dynamic masking policies. Only the users with system administrator or security policy administrator permission can access this view.

Name	Type	Description
polname	name	Name of the masking policy
polenabled	boolean	Specifies whether to enable the masking policy.
maskaction	name	Masking function
labelname	name	Name of the label to which the masking function applies.
masking_object	text	Masking database resource object
filter_name	text	Logical expression of a filter criterion

13.3.73 GS_MATVIEWS

GS_MATVIEWS provides information about each materialized view in the database.

Table 13-183 GS_MATVIEWS columns

Name	Type	Reference	Description
schemaname	name	**PG_NAMESPACE**.nspname	Name of the schema of a materialized view
matviewname	name	**PG_CLASS**.relname	Name of a materialized view
matviewowner	name	**PG_AUTHID**.Erolname	Owner of a materialized view
tablespace	name	**PG_TABLESPACE**.spcname	Tablespace name of a materialized view. If the default tablespace of the database is used, the value is null.
hasindexes	boolean	-	This column is true if a materialized view has (or has recently had) any indexes.
definition	text	-	Definition of a materialized view (a reconstructed SELECT query)

13.3.74 GS_OS_RUN_INFO

GS_OS_RUN_INFO displays the running status of the OS.

Table 13-184 GS_OS_RUN_INFO columns

Name	Type	Description
id	integer	ID
name	text	Name of the OS running status
value	numeric	Value of the OS running status
comments	text	Remarks of the OS running status
cumulative	boolean	Whether the value of the OS running status is cumulative

13.3.75 GS_REDO_STAT

GS_REDO_STAT displays statistics on the replay of session thread logs.

Table 13-185 GS_REDO_STAT columns

Name	Type	Description
phywrts	bigint	Number of times that data is written during log replay
phyblkwrt	bigint	Number of data blocks written during log replay
writetim	bigint	Total time required for writing data during log replay
avgiotim	bigint	Average time required for writing data during log replay
lstiotim	bigint	Time consumed by the last data write operation during log replay
miniotim	bigint	Minimum time consumed by a single data write operation during log replay
maxiowtm	bigint	Maximum time consumed by a single data write operation during log replay

13.3.76 GS_SESSION_CPU_STATISTICS

GS_SESSION_CPU_STATISTICS shows load management information about CPU usage of ongoing complex jobs executed by the current user. The current feature is a lab feature. Contact Huawei technical support before using it.

Table 13-186 GS_SESSION_CPU_STATISTICS columns

Name	Type	Description
datid	oid	OID of the database that the backend is connected to
usename	name	Name of the user logged in to the backend
pid	bigint	Thread ID of the backend
start_time	timestamp with time zone	Time when the statement starts to be executed
min_cpu_time	bigint	Minimum CPU time of the statement across the database nodes, in ms

Name	Type	Description
max_cpu_time	bigint	Maximum CPU time of the statement across the database nodes, in ms
total_cpu_time	bigint	Total CPU time of the statement across the database nodes, in ms
query	text	Statement being executed
node_group	text	Unsupported currently
top_cpu_dn	text	CPU usage

13.3.77 GS_SESSION_MEMORY

GS_SESSION_MEMORY collects statistics about memory usage at the session level, including all the memory allocated to GaussDB and Stream threads on DNs for jobs currently executed by users. If the GUC parameter **enable_memory_limit** is set to **off**, this view is unavailable.

Table 13-187 GS_SESSION_MEMORY columns

Name	Type	Description
sessid	text	Thread start time and ID
init_mem	integer	Memory allocated to the currently executed jobs before they enter the executor, in MB
used_mem	integer	Memory allocated to the currently executed jobs, in MB
peak_mem	integer	Peak memory allocated to the currently executed jobs, in MB

13.3.78 GS_SESSION_MEMORY_CONTEXT

GS_SESSION_MEMORY_CONTEXT displays statistics on memory usage of all sessions based on the MemoryContext node. This view is valid only when **enable_thread_pool** is set to **on**. If the GUC parameter **enable_memory_limit** is set to **off**, this view is unavailable.

The memory context **TempSmallContextGroup** collects information about all memory contexts whose value in the **totalsize** column is less than 8192 bytes in the current thread, and the number of the collected memory contexts is recorded in the **usedsize** column. Therefore, the **totalsize** and **freesize** columns for **TempSmallContextGroup** in the view display the corresponding information about all the memory contexts whose value in the **totalsize** column is less than 8192 bytes in the current thread, and the **usedsize** column displays the number of these memory contexts.

Table 13-188 GS_SESSION_MEMORY_CONTEXT columns

Name	Type	Description
sessid	text	Session start time + session ID (character string: *timestamp.sessionid*)
threadid	bigint	ID of the thread bound to a session (**-1** if no thread is bound)
contextname	text	Name of the memory context
level	smallint	Hierarchy of the memory context
parent	text	Name of the parent memory context
totalsize	bigint	Total size of the memory context, in bytes
freesize	bigint	Total size of released memory in the memory context, in bytes
usedsize	bigint	Size of used memory in the memory context, in bytes. For **TempSmallContextGroup**, this parameter specifies the number of collected memory contexts.

13.3.79 GS_SESSION_MEMORY_DETAIL

GS_SESSION_MEMORY_DETAIL collects statistics about thread memory usage by the memory context. When **enable_thread_pool** is set to **on**, this view contains memory usage of all threads and sessions. If the GUC parameter **enable_memory_limit** is set to **off**, this view is unavailable.

The memory context **TempSmallContextGroup** collects information about all memory contexts whose value in the **totalsize** column is less than 8192 bytes in the current thread, and the number of the collected memory contexts is recorded in the **usedsize** column. Therefore, the **totalsize** and **freesize** columns for **TempSmallContextGroup** in the view display the corresponding information about all the memory contexts whose value in the **totalsize** column is less than 8192 bytes in the current thread, and the **usedsize** column displays the number of these memory contexts.

You can run the **SELECT * FROM gs_session_memctx_detail (***threadid***,'');** statement to record information about all memory contexts of a thread into the *threadid_timestamp***.log** file in the *$GAUSSLOG*/pg_log/${node_name}/dumpmem directory. *threadid* can be obtained from **sessid** in the following table.

Table 13-189 GS_SESSION_MEMORY_DETAIL columns

Name	Type	Description
sessid	text	1. When the thread pool is disabled (**enable_thread_pool = off**), this column indicates the thread start time + session ID (string: **timestamp.sessionid**). 2. When the thread pool is enabled (**enable_thread_pool = on**): If the memory context is at the thread level, this column indicates the thread start time + thread ID (string: **timestamp.threadid**). If the memory context is at the session level, the column indicates the thread start time + session ID (string: **timestamp.sessionid**).
sesstype	text	Thread name
contextname	text	Name of the memory context
level	smallint	Hierarchy of the memory context
parent	text	Name of the parent memory context
totalsize	bigint	Total size of the memory context, in bytes
freesize	bigint	Total size of released memory in the memory context, in bytes
usedsize	bigint	Size of used memory in the memory context, in bytes. For **TempSmallContextGroup**, this parameter specifies the number of collected memory contexts.

13.3.80 GS_SESSION_MEMORY_STATISTICS

GS_SESSION_MEMORY_STATISTICS displays load management information about memory usage of ongoing complex jobs executed by the current user. The current feature is a lab feature. Contact Huawei technical support before using it.

Table 13-190 GS_SESSION_MEMORY_STATISTICS columns

Name	Type	Description
datid	oid	OID of the database that the backend is connected to
usename	name	Name of the user logged in to the backend
pid	bigint	Thread ID of the backend

Name	Type	Description
start_time	timestamp with time zone	Time when the statement starts to be executed
min_peak_memory	integer	Minimum memory peak of the statement across the database nodes, in MB
max_peak_memory	integer	Maximum memory peak of the statement across the database nodes, in MB
spill_info	text	Information about statement spill to the database nodes **None**: The statement has not been spilled to disks on the database nodes. **All**: The statement has been spilled to disks on the database nodes. *[a:b]*: The statement has been spilled to disks on *a* of *b* database nodes.
query	text	Statement being executed
node_group	text	Unsupported currently
top_mem_dn	text	Memory usage

13.3.81 GS_SESSION_STAT

GS_SESSION_STAT collects statistics about session states based on session threads or the **AutoVacuum** thread.

Table 13-191 GS_SESSION_STAT columns

Name	Type	Description
sessid	text	Thread ID and start time
statid	integer	Statistics ID
statname	text	Name of the statistics session
statunit	text	Unit of the statistics session
value	bigint	Value of the statistics session

13.3.82 GS_SESSION_TIME

GS_SESSION_TIME collects statistics about the running time of session threads and time consumed in each execution phase.

Table 13-192 GS_SESSION_TIME columns

Name	Type	Description
sessid	text	Thread ID and start time
stat_id	integer	Statistics ID
stat_name	text	Session type
value	bigint	Session value

13.3.83 GS_SQL_COUNT

GS_SQL_COUNT displays statistics about five types of running statements (**SELECT**, **INSERT**, **UPDATE**, **DELETE**, and **MERGE INTO**) on the current node of the database.

- When a common user queries the **GS_SQL_COUNT** view, statistics about the current node of the user are displayed. When an administrator queries the **GS_SQL_COUNT** view, statistics about the current node of all users are displayed.
- When the database or the node is restarted, the statistics are cleared and will be measured again.
- The system counts when a node receives a query, including a query inside the database.

Table 13-193 GS_SQL_COUNT columns

Name	Type	Description
node_name	name	Node name
user_name	name	Username
select_count	bigint	Statistical result of the **SELECT** statement
update_count	bigint	Statistical result of the **UPDATE** statement
insert_count	bigint	Statistical result of the **INSERT** statement
delete_count	bigint	Statistical result of the **DELETE** statement
mergeinto_count	bigint	Statistical result of the **MERGE INTO** statement
ddl_count	bigint	Number of DDL statements
dml_count	bigint	Number of DML statements

Name	Type	Description
dcl_count	bigint	Number of DML statements
total_select_elapse	bigint	Total response time of **SELECT** statements (unit: μs)
avg_select_elapse	bigint	Average response time of **SELECT** statements (unit: μs)
max_select_elapse	bigint	Maximum response time of **SELECT** statements (unit: μs)
min_select_elapse	bigint	Minimum response time of **SELECT** statements (unit: μs)
total_update_elapse	bigint	Total response time of **UPDATE** statements (unit: μs)
avg_update_elapse	bigint	Average response time of **UPDATE** statements (unit: μs)
max_update_elapse	bigint	Maximum response time of **UPDATE** statements (unit: μs)
min_update_elapse	bigint	Minimum response time of **UPDATE** statements (unit: μs)
total_insert_elapse	bigint	Total response time of **INSERT** statements (unit: μs)
avg_insert_elapse	bigint	Average response time of **INSERT** statements (unit: μs)
max_insert_elapse	bigint	Maximum response time of **INSERT** statements (unit: μs)
min_insert_elapse	bigint	Minimum response time of **INSERT** statements (unit: μs)
total_delete_elapse	bigint	Total response time of **DELETE** statements (unit: μs)
avg_delete_elapse	bigint	Average response time of **DELETE** statements (unit: μs)
max_delete_elapse	bigint	Maximum response time of **DELETE** statements (unit: μs)
min_delete_elapse	bigint	Minimum response time of **DELETE** statements (unit: μs)

13.3.84 GS_STAT_SESSION_CU

GS_STAT_SESSION_CU queries the CU hit rate of running sessions on each node of the entire database. The data about a session is cleared when you exit this session or restart the database.

Table 13-194 GS_STAT_SESSION_CU columns

Name	Type	Description
mem_hit	integer	Number of memory hits
hdd_sync_read	integer	Number of synchronous hard disk reads
hdd_asyn_read	integer	Number of asynchronous hard disk reads

13.3.85 GS_THREAD_MEMORY_CONTEXT

GS_THREAD_MEMORY_CONTEXT displays statistics about memory usage of all threads based on MemoryContext nodes. This view is equivalent to the **GS_SESSION_MEMORY_DETAIL** view when **enable_thread_pool** is set to **off**. If the GUC parameter **enable_memory_limit** is set to **off**, this view is unavailable.

The memory context **TempSmallContextGroup** collects information about all memory contexts whose value in the **totalsize** column is less than 8192 bytes in the current thread, and the number of the collected memory contexts is recorded in the **usedsize** column. Therefore, the **totalsize** and **freesize** columns for **TempSmallContextGroup** in the view display the corresponding information about all the memory contexts whose value in the **totalsize** column is less than 8192 bytes in the current thread, and the **usedsize** column displays the number of these memory contexts.

Table 13-195 GS_THREAD_MEMORY_CONTEXT columns

Name	Type	Description
threadid	text	Thread start time + thread ID (string: *timestamp.sessionid*)
tid	bigint	Thread ID
thrdtype	text	Thread type
contextname	text	Name of the memory context
level	smallint	Hierarchy of the memory context
parent	text	Name of the parent memory context
totalsize	bigint	Total size of the memory context, in bytes

Name	Type	Description
freesize	bigint	Total size of released memory in the current memory context, in bytes
usedsize	bigint	Size of used memory in the memory context, in bytes. For **TempSmallContextGroup**, this parameter specifies the number of collected memory contexts.

13.3.86 GS_TOTAL_MEMORY_DETAIL

GS_TOTAL_MEMORY_DETAIL collects statistics about memory usage of the current database node in the unit of MB. If the GUC parameter **enable_memory_limit** is set to **off**, this view is unavailable.

Table 13-196 GS_TOTAL_MEMORY_DETAIL columns

Name	Type	Description
nodename	text	Node name

Name	Type	Description
memorytype	text	Memory type. The value must be one of the following: ● **max_process_memory**: memory occupied by the GaussDB instance ● **process_used_memory**: memory occupied by the GaussDB process ● **max_dynamic_memory**: maximum dynamic memory ● **dynamic_used_memory**: used dynamic memory ● **dynamic_peak_memory**: dynamic peak value of the memory ● **dynamic_used_shrctx**: maximum dynamic shared memory context ● **dynamic_peak_shrctx**: dynamic peak value of the shared memory context ● **max_shared_memory**: maximum shared memory ● **shared_used_memory**: used shared memory ● **max_cstore_memory**: maximum memory allowed for the column ● **cstore_used_memory**: memory used in column storage ● **max_sctpcomm_memory**: maximum memory allowed for the communication library ● **sctpcomm_used_memory**: memory used by the communication library ● **sctpcomm_peak_memory**: memory peak of the communication library ● **other_used_memory**: other used memory
memorymbytes	integer	Size of allocated memory-typed memory

13.3.87 GS_TOTAL_NODEGROUP_MEMORY_DETAIL

GS_TOTAL_NODEGROUP_MEMORY_DETAIL returns the memory usage (in MB) of the current logical instance of the database. If **enable_memory_limit** is set to **off**, this function cannot be used.

Table 13-197 GS_TOTAL_NODEGROUP_MEMORY_DETAIL columns

Name	Type	Description
ngname	text	Logical instance name
memorytype	text	Memory type. The value must be one of the following: ● **ng_total_memory**: total memory of the logical instance ● **ng_used_memory**: memory usage of the logical instance ● **ng_estimate_memory**: estimated memory usage of the logical instance ● **ng_foreignrp_memsize**: total memory of the external resource pool of the logical instance ● **ng_foreignrp_usedsize**: memory usage of the external resource pool of the logical instance ● **ng_foreignrp_peaksize**: peak memory usage of the external resource pool of the logical instance ● **ng_foreignrp_mempct**: percentage of the external resource pool of the logical instance to the total memory of the logical instance ● **ng_foreignrp_estmsize**: estimated memory usage of the external resource pool of the logical instance
memorymbytes	integer	Size of allocated memory-typed memory

13.3.88 GS_WLM_CGROUP_INFO

GS_WLM_CGROUP_INFO displays information about a Cgroup for a job that is being executed.

Table 13-198 GS_WLM_CGROUP_INFO columns

Name	Type	Description
cgroup_name	text	Cgroup name
priority	integer	Priority of the job
usage_percent	integer	Percentage of resources used by the Cgroup
shares	bigint	CPU quota allocated to the Cgroup
cpuacct	bigint	Allocated CPU quota
cpuset	text	Allocated CPU cores

Name	Type	Description
relpath	text	Relative path of the Cgroup
valid	text	Whether the Cgroup is valid
node_group	text	Name of the logical database instance

13.3.89 GS_WLM_EC_OPERATOR_STATISTICS

GS_WLM_EC_OPERATOR_STATISTICS displays operators of the Extension Connector jobs that are being executed by the current user. To query the view, you must have the sysadmin permission. The current feature is a lab feature. Contact Huawei technical support before using it.

Table 13-199 GS_WLM_EC_OPERATOR_STATISTICS columns

Name	Type	Description
queryid	bigint	Internal query ID used for Extension Connector statement execution
plan_node_id	integer	Plan node ID of the execution plan of an Extension Connector operator
start_time	timestamp with time zone	Time when the Extension Connector operator starts to process the first data record
ec_status	text	Status of the Extension Connector job ● **EC_STATUS_INIT**: initialized ● **EC_STATUS_CONNECTED**: connected ● **EC_STATUS_EXECUTED**: executed ● **EC_STATUS_FETCHING**: fetching ● **EC_STATUS_END**: ended
ec_execute_datanode	text	Name of the DN executing an Extension Connector job
ec_dsn	text	DSN used by an Extension Connector job
ec_username	text	Username used by the Extension Connector job to access a remote database instance (the value is null if the remote database instance is SPARK.)
ec_query	text	Statement sent by an Extension Connector job to a remote database instance

Name	Type	Description
ec_libodbc_type	text	Type of the unixODBC driver used by an Extension Connector job ● Type 1: corresponds to **libodbc.so.1**. ● Type 2: corresponds to **libodbc.so.2**.
ec_fetch_count	bigint	Number of data records processed by an Extension Connector job

13.3.90 GS_WLM_OPERATOR_HISTORY

GS_WLM_OPERATOR_HISTORY displays records of operators in jobs that have been executed by the current user on the current primary database node. Only the user with sysadmin permission can query this view.

Columns in this view are the same as those in **Table 13-32**.

13.3.91 GS_WLM_OPERATOR_STATISTICS

GS_WLM_OPERATOR_STATISTICS displays the operators of the jobs that are being executed by the current user. Only users with the **sysadmin** permission can query this view.

Table 13-200 GS_WLM_OPERATOR_STATISTICS columns

Name	Type	Description
queryid	bigint	Internal query ID used for statement execution
pid	bigint	Thread ID of the backend
plan_node_id	integer	Plan node ID of the execution plan
plan_node_name	text	Name of the operator corresponding to the plan node ID
start_time	timestamp with time zone	Time when the operator starts to process the first data record
duration	bigint	Total execution time of the operator, in ms
status	text	Execution status of the current operator, which can be **finished** or **running**
query_dop	integer	DOP of the operator
estimated_rows	bigint	Number of rows estimated by the optimizer

Name	Type	Description
tuple_processed	bigint	Number of elements returned by the operator
min_peak_memory	integer	Minimum peak memory used by the operator on database instances (unit: MB)
max_peak_memory	integer	Maximum peak memory used by the operator on database instances (unit: MB)
average_peak_memory	integer	Average peak memory used by the operator on database instances (unit: MB)
memory_skew_percent	integer	Memory usage skew of the operator among database instances
min_spill_size	integer	Minimum spilled data among database instances when a spill occurs (unit: MB). The default value is **0**.
max_spill_size	integer	Maximum spilled data among database instances when a spill occurs (unit: MB). The default value is **0**.
average_spill_size	integer	Average spilled data among database instances when a spill occurs (unit: MB). The default value is **0**.
spill_skew_percent	integer	Database instance spill skew when a spill occurs
min_cpu_time	bigint	Minimum execution time of the operator on database instances (unit: ms)
max_cpu_time	bigint	Maximum execution time of the operator on database instances (unit: ms)
total_cpu_time	bigint	Total execution time of the operator on database instances (unit: ms)
cpu_skew_percent	integer	Execution time skew among database instances

Name	Type	Description
warning	text	Warning. The following warnings are displayed: ● Sort/SetOp/HashAgg/HashJoin spill ● Spill file size large than 256MB ● Broadcast size large than 100MB ● Early spill ● Spill times is greater than 3 ● Spill on memory adaptive ● Hash table conflict

13.3.92 GS_WLM_PLAN_OPERATOR_HISTORY

GS_WLM_PLAN_OPERATOR_HISTORY displays the execution plan operator-level records of the current user on the primary database node after the job is complete.

Columns in this view are the same as those in **Table 13-34**.

13.3.93 GS_WLM_REBUILD_USER_RESOURCE_POOL

GS_WLM_REBUILD_USER_RESOURCE_POOL is used to rebuild a user's resource pool information in memory on the current connection node, with no output. This view is only used as a remedy when resource pool information is missing or misplaced. Only the user with sysadmin permission can query this view.

Table 13-201 GS_WLM_REBUILD_USER_RESOURCE_POOL columns

Name	Type	Description
gs_wlm_rebuild_user_resource_pool	Boolean	Rebuilds information about the user resource pool in the memory. **t** indicates success, and **f** indicates failure.

13.3.94 GS_WLM_RESOURCE_POOL

GS_WLM_RESOURCE_POOL records statistics on a resource pool.

Table 13-202 GS_WLM_RESOURCE_POOL columns

Name	Type	Description
rpoid	oid	OID of the resource pool
respool	name	Name of the resource pool

Name	Type	Description
control_group	name	Unsupported currently
parentid	oid	OID of the parent resource pool
ref_count	integer	Number of jobs associated with the resource pool
active_points	integer	Number of used points in the resource pool
running_count	integer	Number of jobs running in the resource pool
waiting_count	integer	Number of jobs queuing in the resource pool
io_limits	integer	IOPS upper limit of the resource pool
io_priority	integer	I/O priority of the resource pool

13.3.95 GS_WLM_SESSION_HISTORY

GS_WLM_SESSION_HISTORY displays load management information about all completed jobs executed by the current user on database instances. The current feature is a lab feature. Contact Huawei technical support before using it. Only users with the **sysadmin** or **monitor admin** permission can query this view.

Table 13-203 GS_WLM_SESSION_HISTORY columns

Name	Type	Description
datid	oid	OID of the database this backend is connected to.
dbname	text	Name of the database the backend is connected to.
schemaname	text	Schema name.
nodename	text	Name of the database instance where the statement is executed.
username	text	Username used for connecting to the backend.
application_name	text	Name of the application connected to the backend.
client_addr	inet	IP address of the client connected to the backend. If this column is null, it indicates either the client is connected via a Unix socket on the server or this is an internal process, such as autovacuum.

Name	Type	Description
client_hostname	text	Host name of the connected client, as reported by a reverse DNS lookup of **client_addr**. This column will be non-null only for IP connections and only when **log_hostname** is enabled.
client_port	integer	TCP port number that the client uses for communication with this backend (**–1** if a Unix socket is used)
query_band	text	Job type, which is specified by the GUC parameter **query_band**. The default value is a null string.
block_time	bigint	Duration that the statement is blocked before being executed, including the statement parsing and optimization duration (unit: ms)
start_time	timestamp with time zone	Time when the statement starts to be run.
finish_time	timestamp with time zone	Time when the statement execution ends.
duration	bigint	Execution time of a statement. The unit is ms.
estimate_total_time	bigint	Estimated execution time of a statement. The unit is ms.
status	text	Final statement execution status, which can be **finished** (normal) or **aborted** (abnormal).
abort_info	text	Exception information displayed if the final statement execution status is **aborted**.
resource_pool	text	Resource pool used by the user.
control_group	text	Cgroup used by the statement.
estimate_memory	integer	Estimated memory size of the statement.
min_peak_memory	integer	Minimum memory peak of the statement across the database instances, in MB.
max_peak_memory	integer	Maximum memory peak of the statement across the database instances, in MB.
average_peak_memory	integer	Average memory usage during statement execution. The unit is MB.
memory_skew_percent	integer	Memory usage skew of the statement among the database instances.

Name	Type	Description
spill_info	text	Information about statement spill to the database instances: ● **None**: The statement has not been spilled to disks on the database instances. ● **All**: The statement has been spilled to disks on the database instances. ● *[a:b]*: The statement has been spilled to disks on *a* of *b* database instances.
min_spill_size	integer	Minimum spilled data among database instances when a spill occurs (unit: MB). Default value: **0**
max_spill_size	integer	Maximum spilled data among database instances when a spill occurs (unit: MB). Default value: **0**
average_spill_size	integer	Average spilled data among database instances when a spill occurs (unit: MB). Default value: **0**
spill_skew_percent	integer	Database instance spill skew when a spill occurs.
min_dn_time	bigint	Minimum execution time of the statement across the database instances (unit: ms).
max_dn_time	bigint	Maximum execution time of the statement across the database instances (unit: ms).
average_dn_time	bigint	Average execution time of the statement across the database instances (unit: ms).
dntime_skew_percent	integer	Execution time skew of the statement among the database instances.
min_cpu_time	bigint	Minimum CPU time of the statement across the database instances (unit: ms).
max_cpu_time	bigint	Maximum CPU time of the statement across the database instances (unit: ms).
total_cpu_time	bigint	Total CPU time of the statement across the database instances (unit: ms).
cpu_skew_percent	integer	CPU time skew of the statement among the database instances.
min_peak_iops	integer	Minimum IOPS peak of the statement across the database instances. It is counted by ones in a column-store table and by ten thousands in a row-store table.

Name	Type	Description
max_peak_iops	integer	Maximum IOPS peak of the statement across the database instances. It is counted by ones in a column-store table and by ten thousands in a row-store table.
average_peak_iops	integer	Average IOPS peak of the statement across the database instances. It is counted by ones in a column-store table and by ten thousands in a row-store table.
iops_skew_percent	integer	I/O skew of the statement among the database instances.
warning	text	Warning. The following warnings are displayed: ● Spill file size large than 256MB ● Broadcast size large than 100MB ● Early spill ● Spill times is greater than 3 ● Spill on memory adaptive ● Hash table conflict
queryid	bigint	Internal query ID used for statement execution.
query	text	Statement executed.
query_plan	text	Execution plan of a statement.
node_group	text	Logical database instance of the user to which the statement belongs.
cpu_top1_node_name	text	Name of the node with the 1st CPU usage.
cpu_top2_node_name	text	Name of the node with the 2nd CPU usage.
cpu_top3_node_name	text	Name of the node with the 3rd CPU usage.
cpu_top4_node_name	text	Name of the node with the 4th CPU usage.
cpu_top5_node_name	text	Name of the node with the 5th CPU usage.
mem_top1_node_name	text	Name of the node with the 1st memory usage.
mem_top2_node_name	text	Name of the node with the 2nd memory usage.

Name	Type	Description
mem_top3_node_name	text	Name of the node with the 3rd memory usage.
mem_top4_node_name	text	Name of the node with the 4th memory usage.
mem_top5_node_name	text	Name of the node with the 5th memory usage.
cpu_top1_value	bigint	CPU usage.
cpu_top2_value	bigint	CPU usage.
cpu_top3_value	bigint	CPU usage.
cpu_top4_value	bigint	CPU usage.
cpu_top5_value	bigint	CPU usage.
mem_top1_value	bigint	Memory usage.
mem_top2_value	bigint	Memory usage.
mem_top3_value	bigint	Memory usage.
mem_top4_value	bigint	Memory usage.
mem_top5_value	bigint	Memory usage.
top_mem_dn	text	Top *N* memory usage.
top_cpu_dn	text	Top *N* CPU usage.

13.3.96 GS_WLM_SESSION_INFO

GS_WLM_SESSION_INFO displays load management information about a completed job executed on the database instance. The current feature is a lab feature. Contact Huawei technical support before using it. Only the user with sysadmin permission can query this view.

For details about the columns, see **Table 13-203**.

13.3.97 GS_WLM_SESSION_INFO_ALL

GS_WLM_SESSION_INFO_ALL displays load management information about a completed job executed on the database instance. (The current feature is a lab feature. Contact Huawei technical support before using it.) Only the user with the **sysadmin** or **monitor admin** permission can query this view.

Table 13-204 GS_WLM_SESSION_INFO_ALL columns

Name	Type	Description
datid	oid	OID of the database that the backend is connected to
dbname	text	Name of the database that the backend is connected to
schemaname	text	Schema name
nodename	text	Name of the CN where the statement is executed
username	text	Username used for connecting to the backend
application_name	text	Name of the application connected to the backend
client_addr	inet	IP address of the client connected to the backend. If this column is null, it indicates either the client is connected via a Unix socket on the server or this is an internal process, such as autovacuum.
client_hostname	text	Host name of the connected client, as reported by a reverse DNS lookup of **client_addr**. This column will be non-null only for IP connections and only when **log_hostname** is enabled.
client_port	integer	TCP port number that the client uses for communication with this backend (**-1** if a Unix socket is used)
query_band	text	Job type, which is specified by the GUC parameter **query_band**. The default value is a null string.
block_time	bigint	Duration that the statement is blocked before being executed, including the statement parsing and optimization duration (unit: ms)
start_time	timestamp with time zone	Time when the statement starts to be executed
finish_time	timestamp with time zone	Time when the statement execution ends
duration	bigint	Execution time of the statement (unit: ms)
estimate_total_time	bigint	Estimated execution time of the statement (unit: ms)

Name	Type	Description
status	text	Final statement execution status, which can be **finished** (normal) or **aborted** (abnormal).
abort_info	text	Exception information displayed if the final statement execution status is **aborted**
resource_pool	text	Resource pool used by the user
control_group	text	Cgroup used by the statement
estimate_memory	integer	Estimated memory size of the statement.
min_peak_memory	integer	Minimum peak memory of the statement across all DNs (unit: MB)
max_peak_memory	integer	Maximum peak memory of the statement across all DNs (unit: MB)
average_peak_memory	integer	Average memory usage during statement execution (unit: MB)
memory_skew_percent	integer	Memory usage skew of the statement among DNs
spill_info	text	Statement spill information on all DNs. ● **None**: No data is spilled to disks. ● **All**: Data is spilled to disks on all DNs. ● **[*a*:*b*]**: The statement has been spilled to disks on *a* of *b* DNs.
min_spill_size	integer	Minimum spilled data among all DNs when a spill occurs (unit: MB; default value: **0**)
max_spill_size	integer	Maximum spilled data among all DNs when a spill occurs (unit: MB; default value: **0**)
average_spill_size	integer	Average spilled data among all DNs when a spill occurs (unit: MB; default value: **0**)
spill_skew_percent	integer	DN spill skew when a spill occurs
min_dn_time	bigint	Minimum execution time of the statement across all DNs (unit: ms)
max_dn_time	bigint	Maximum execution time of the statement across all DNs (unit: ms)
average_dn_time	bigint	Average execution time of the statement across all DNs (unit: ms)
dntime_skew_percent	integer	Execution time skew of the statement among DNs

Name	Type	Description
min_cpu_time	bigint	Minimum CPU time of the statement across all DNs (unit: ms)
max_cpu_time	bigint	Maximum CPU time of the statement across all DNs (unit: ms)
total_cpu_time	bigint	Total CPU time of the statement across all DNs (unit: ms)
cpu_skew_percent	integer	CPU time skew of the statement among DNs
min_peak_iops	integer	Minimum peak IOPS of the statement across all DNs. It is counted by ones in a column-store table and by ten thousands in a row-store table.
max_peak_iops	integer	Maximum peak IOPS of the statement across all DNs. It is counted by ones in a column-store table and by ten thousands in a row-store table.
average_peak_iops	integer	Average peak IOPS of the statement across all DNs. It is counted by ones in a column-store table and by ten thousands in a row-store table.
iops_skew_percent	integer	I/O skew of the statement among DNs
warning	text	Warning. The following warnings and warnings related to **Optimizing SQL Self-Diagnosis** are displayed: ● Spill file size large than 256MB ● Broadcast size large than 100MB ● Early spill ● Spill times is greater than 3 ● Spill on memory adaptive ● Hash table conflict
queryid	bigint	Internal query ID used for statement execution
query	text	Statement executed
query_plan	text	Execution plan of the statement
node_group	text	Logical cluster of the user running the statement (The current feature is a lab feature. Contact Huawei technical support before using it.)

Name	Type	Description
cpu_top1_node_name	text	Name of the node with the highest CPU usage
cpu_top2_node_name	text	Name of the node with the second highest CPU usage
cpu_top3_node_name	text	Name of the node with the third highest CPU usage
cpu_top4_node_name	text	Name of the node with the fourth highest CPU usage
cpu_top5_node_name	text	Name of the node with the fifth highest CPU usage
mem_top1_node_name	text	Name of the node with the highest memory usage
mem_top2_node_name	text	Name of the node with the second highest memory usage
mem_top3_node_name	text	Name of the node with the third highest memory usage
mem_top4_node_name	text	Name of the node with the fourth highest memory usage
mem_top5_node_name	text	Name of the node with the fifth highest memory usage
cpu_top1_value	bigint	CPU usage
cpu_top2_value	bigint	CPU usage
cpu_top3_value	bigint	CPU usage
cpu_top4_value	bigint	CPU usage
cpu_top5_value	bigint	CPU usage
mem_top1_value	bigint	Memory usage
mem_top2_value	bigint	Memory usage
mem_top3_value	bigint	Memory usage
mem_top4_value	bigint	Memory usage
mem_top5_value	bigint	Memory usage
top_mem_dn	text	Top N memory usage

Name	Type	Description
top_cpu_dn	text	Top N CPU usage
n_returned_rows	bigint	Number of rows in the result set returned by the **SELECT** statement.
n_tuples_fetched	bigint	Number of rows randomly scanned.
n_tuples_returned	bigint	Number of rows sequentially scanned.
n_tuples_inserted	bigint	Number of rows inserted.
n_tuples_updated	bigint	Number of rows updated.
n_tuples_deleted	bigint	Number of rows deleted.
n_blocks_fetched	bigint	Number of buffer block access times.
n_blocks_hit	bigint	Number of buffer block hits.
db_time	bigint	Valid DB time, which is accumulated if multiple threads are involved (unit: μs).
cpu_time	bigint	CPU time (unit: μs).
execution_time	bigint	Execution time in the executor (unit: μs).
parse_time	bigint	SQL parsing time (unit: μs).
plan_time	bigint	SQL plan generation time (unit: μs).
rewrite_time	bigint	SQL rewriting time (unit: μs).
pl_execution_time	bigint	Execution time of PL/pgSQL (unit: μs).
pl_compilation_time	bigint	Compilation time of PL/pgSQL (unit: μs).
net_send_time	bigint	Network time (unit: μs).
data_io_time	bigint	I/O time (unit: μs).
is_slow_query	bigint	Whether the record is a slow SQL record.

13.3.98 GS_WLM_SESSION_STATISTICS

GS_WLM_SESSION_STATISTICS displays load management information about jobs being executed by the current user on the database instance. The current feature is a lab feature. Contact Huawei technical support before using it. Only users with the **sysadmin** permission can query this view.

Table 13-205 GS_WLM_SESSION_STATISTICS columns

Name	Type	Description
datid	oid	OID of the database the backend is connected to
dbname	name	Name of the database the backend is connected to
schemaname	text	Schema name
nodename	text	Name of the database instance where the statement is executed
username	name	Username used for connecting to the backend
application_name	text	Name of the application connected to the backend
client_addr	inet	IP address of the client connected to the backend. If this column is null, it indicates either that the client is connected via a Unix socket on the server machine or that this is an internal process, such as autovacuum.
client_hostname	text	Host name of the connected client, as reported by a reverse DNS lookup of **client_addr**. This column will be non-null only for IP connections and only when **log_hostname** is enabled.
client_port	integer	TCP port number that the client uses for communication with the backend (**–1** if a Unix socket is used)
query_band	text	Job type, which is specified by the GUC parameter **query_band**. The default value is a null string.
pid	bigint	ID of the backend thread
sessionid	bigint	Session ID
block_time	bigint	Block time before the statement is run. The unit is ms.
start_time	timestamp with time zone	Time when the statement execution starts
duration	bigint	Duration that the statement has been executed (unit: ms)

Name	Type	Description
estimate_total_time	bigint	Estimated execution time of the statement. The unit is ms.
estimate_left_time	bigint	Estimated remaining time of statement execution. The unit is ms.
enqueue	text	Resource status in workload management (The current feature is a lab feature. Contact Huawei technical support before using it.)
resource_pool	name	Resource pool used by the user
control_group	text	Cgroup used by the statement
estimate_memory	integer	Estimated memory used by the statement. The unit is MB.
min_peak_memory	integer	Minimum memory peak of the statement across the database instances, in MB
max_peak_memory	integer	Maximum memory peak of the statement across the database instances, in MB
average_peak_memory	integer	Average memory usage during statement execution, in MB
memory_skew_percent	integer	Memory usage skew of the statement among the database instances
spill_info	text	Information about statement spill to the database instances: ● **None**: The statement has not been spilled to disks on the database instances. ● **All**: The statement has been spilled to disks on the database instances. ● *[a:b]*: The statement has been spilled to disks on *a* of *b* database instances.
min_spill_size	integer	Minimum spilled data among database instances when a spill occurs (unit: MB). The default value is **0**.
max_spill_size	integer	Maximum spilled data among database instances when a spill occurs (unit: MB). The default value is **0**.
average_spill_size	integer	Average spilled data among database instances when a spill occurs (unit: MB). The default value is **0**.
spill_skew_percent	integer	Database instance spill skew when a spill occurs

Name	Type	Description
min_dn_time	bigint	Minimum execution time of the statement across the database instances (unit: ms)
max_dn_time	bigint	Maximum execution time of the statement across the database instances (unit: ms)
average_dn_time	bigint	Average execution time of the statement across the database instances (unit: ms)
dntime_skew_percent	integer	Execution time skew of the statement among the database instances
min_cpu_time	bigint	Minimum CPU time of the statement across the database instances (unit: ms)
max_cpu_time	bigint	Maximum CPU time of the statement across the database instances (unit: ms)
total_cpu_time	bigint	Total CPU time of the statement across the database instances (unit: ms)
cpu_skew_percent	integer	CPU time skew of the statement among the database instances
min_peak_iops	integer	Minimum IOPS peak of the statement across the database instances. It is counted by ones in a column-store table and by ten thousands in a row-store table.
max_peak_iops	integer	Maximum IOPS peak of the statement across the database instances. It is counted by ones in a column-store table and by ten thousands in a row-store table.
average_peak_iops	integer	Average IOPS peak of the statement across the database instances. It is counted by ones in a column-store table and by ten thousands in a row-store table.
iops_skew_percent	integer	I/O skew of the statement among the database instances
warning	text	Warning. The following warnings are displayed: ● Spill file size large than 256MB ● Broadcast size large than 100MB ● Early spill ● Spill times is greater than 3 ● Spill on memory adaptive ● Hash table conflict
queryid	bigint	Internal query ID used for statement execution
query	text	Statement that is being executed

Name	Type	Description
query_plan	text	Execution plan of the statement
node_group	text	Logical database instance of the user to which the statement belongs
top_cpu_dn	text	Top *N* CPU usage
top_mem_dn	text	Top *N* memory usage

13.3.99 GS_WLM_USER_INFO

GS_WLM_USER_INFO displays user statistics.

Table 13-206 GS_WLM_USER_INFO columns

Name	Type	Description
userid	oid	OID of a user
username	name	Username
sysadmin	boolean	Whether the user is the administrator
rpoid	oid	OID of the associated resource pool
respool	name	Name of the associated resource pool
parentid	oid	OID of the user group
totalspace	bigint	Available space limit of the user
spacelimit	bigint	User table space limit
childcount	interger	Number of child users
childlist	text	Child user list

13.3.100 GS_WLM_WORKLOAD_RECORDS

This view is not supported in the centralized system.

13.3.101 GV_SESSION

GV_SESSION displays all session information about the current session.

Table 13-207 GV_SESSION columns

Name	Type	Description
sid	bigint	Name of the database where the hot key is located.
serial#	integer	Name of the schema where the hot key is located.
schemaname	name	Name of the table where the hot key is located.
user#	oid	Value of a hot key.
username	name	Hash value of the hot key in the database. If the table is a list or range distribution table, the value of this field is 0.
machine	text	Frequency of accessing the hot key.
sql_id	bigint	SQL OID.
client_info	text	Client information.
event	text	Queuing status of a statement. The value must be one of the following: ● **waiting in queue**: The statement is in the queue. ● **Empty**: The statement is running.
sql_exec_start	timestamp	SQL statement execution start time.
program	text	Name of the application connected to the backend.
status	text	Overall status of this backend. The value must be one of the following: ● **active**: The backend is executing a query. ● **idle**: The backend is waiting for a new client command. ● **idle in transaction**: The backend is in a transaction, but there is no statement being executed in the transaction. ● **idle in transaction** (aborted): The backend is in a transaction, but there are statements failed in the transaction. ● **fastpath function call**: The backend is executing a fast-path function. ● **disabled**: This state is reported if **track_activities** is disabled in this backend.

13.3.102 MPP_TABLES

The following information is displayed in the **MPP_TABLES** view:

Table 13-208 MPP_TABLES columns

Name	Type	Description
schemaname	name	Name of the schema that contains a table
tablename	name	Table name
tableowner	name	Table owner
tablespace	name	Tablespace containing the table
pgroup	name	Name of a node cluster
nodeoids	oidvector_extend	List of distributed table node OIDs

13.3.103 MY_COL_COMMENTS

MY_COL_COMMENTS displays column comments of the table accessible to the current user. This view exists in both **PG_CATALOG** and **SYS schema**.

Table 13-209 MY_COL_COMMENTS columns

Name	Type	Description
owner	character varying(64)	Constraint creator
owner	character varying(64)	Table owner
table_name	character varying(64)	Table name
column_name	character varying(64)	Column name
comments	text	Comments

13.3.104 MY_CONSTRAINTS

MY_CONSTRAINTS displays table constraint information accessible to the current user. This view exists in both **PG_CATALOG** and **SYS schema**.

Table 13-210 MY_CONSTRAINTS columns

Name	Type	Description
owner	character varyiny(64)	Constraint creator
constraint_name	vcharacter varying(64)	Constraint name
constraint_type	text	Constraint type ● **c**: check constraint ● **f**: foreign key constraint ● **p**: primary key constraint ● **u**: unique constraint
table_name	character varying(64)	Name of a constraint-related table
index_owner	character varying(64)	Owner of a constraint-related index (only for the unique constraint and primary key constraint)
index_name	character varying(64)	Name of a constraint-related index (only for the unique constraint and primary key constraint)

13.3.105 MY_CONS_COLUMNS

MY_CONS_COLUMNS displays information about primary key constraint columns in tables accessible to the current user. This view exists in both **PG_CATALOG** and **SYS schema**.

Table 13-211 MY_CONS_COLUMNS columns

Name	Type	Description
owner	character varying(64)	Constraint creator
table_name	character varying(64)	Name of a constraint-related table
column_name	character varying(64)	Name of a constraint-related column
constraint_name	character varying(64)	Constraint name
position	smallint	Position of a column in a table

13.3.106 MY_INDEXES

MY_INDEXES displays index information in the current schema. This view exists in both **PG_CATALOG** and **SYS schema**.

Table 13-212 MY_INDEXES columns

Name	Type	Description
owner	character varying(64)	Index owner
index_name	character varying(64)	Index name
table_name	character varying(64)	Name of a table corresponding to an index
uniqueness	text	Whether an index is unique
partitioned	character(3)	Whether an index has the property of partitioned tables
generated	character varying(1)	Whether the name of an index is generated by the system

13.3.107 MY_IND_COLUMNS

MY_IND_COLUMNS displays column information about all indexes accessible to the current user. This view exists in both **PG_CATALOG** and **SYS schema**.

Table 13-213 MY_IND_COLUMNS columns

Name	Type	Description
index_owner	character varying(64)	Index owner
index_name	character varying(64)	Index name
table_owner	character varying(64)	Table owner
table_name	character varying(64)	Table name
column_name	name	Column name
column_position	smallint	Position of a column in an index

13.3.108 MY_IND_EXPRESSIONS

MY_IND_EXPRESSIONS displays information about function-based expression indexes accessible to the current user. This view exists in both **PG_CATALOG** and **SYS schema**.

Table 13-214 MY_IND_EXPRESSIONS columns

Name	Type	Description
table_owner	character varying(64)	Table owner
table_name	character varying(64)	Table name
index_owner	character varying(64)	Index owner
index_name	character varying(64)	Index name
column_expression	text	Function-based index expression of a specified column
column_position	smallint	Position of a column in an index

13.3.109 MY_IND_PARTITIONS

MY_IND_PARTITIONS displays the partition information of indexes (excluding global indexes of partitioned tables) accessible to the current user. This view exists in the **PG_CATALOG** and **SYS** schemas.

Table 13-215 MY_IND_PARTITIONS columns

Name	Type	Description
index_owner	character varying(64)	Owner name of the partitioned table index to which an index partition belongs
index_name	character varying(64)	Index name of the partitioned table index to which an index partition belongs
partition_name	character varying(64)	Name of an index partition
def_tablespace_name	name	Tablespace name of an index partition
high_value	text	Limit of the partition corresponding to an index partition **NOTE** ● For range partitioning and interval partitioning, the upper limit of each partition is displayed. ● For list partitioning, the value list of each partition is displayed. ● For hash partitioning, the number of each partition is displayed.

Name	Type	Description
index_partition_usable	boolean	Whether an index partition is available ● **t** (true): yes ● **f** (false): no
schema	character varying(64)	Schema of the partitioned table index to which an index partition belongs
high_value_length	integer	Character length of the limit of the partition corresponding to an index partition

13.3.110 MY_IND_SUBPARTITIONS

MY_IND_SUBPARTITIONS displays information about level-2 partitions of indexes (excluding global indexes of partitioned tables) owned by the current user. This view exists in the **PG_CATALOG** and **SYS** schemas.

Table 13-216 MY_IND_SUBPARTITIONS columns

Name	Type	Description
index_owner	character varying(64)	Owner name of the partitioned table index to which an index partition belongs
index_name	character varying(64)	Index name of the partitioned table index to which an index partition belongs
partition_name	character varying(64)	Name of the partition where an index is located
subpartition_name	character varying(64)	Name of the level-2 partition where an index is located
def_tablespace_name	name	Tablespace name of an index partition

Name	Type	Description
high_value	text	Limit of the partition corresponding to an index partition **NOTE** ● For range partitioning and interval partitioning, the upper limit of each partition is displayed. ● For list partitioning, the value list of each partition is displayed. ● For hash partitioning, the number of each partition is displayed.
index_partition_usable	boolean	Whether an index partition is available ● **t** (true): yes ● **f** (false): no
schema	character varying(64)	Schema of the partitioned table index to which an index partition belongs
high_value_length	integer	Character length of the limit of the partition corresponding to an index partition

13.3.111 MY_JOBS

MY_JOBS displays all jobs owned by the user. This view exists in the **PG_CATALOG** and **SYS** schemas.

Table 13-217 MY_JOBS columns

Name	Type	Description
job	bigint	Job ID
log_user	name	Username of a job creator
priv_user	name	Username of a job executor
dbname	name	Name of the database in which a job is created
start_date	timestamp without time zone	Job start time

Name	Type	Description
start_suc	text	Start time of a successful job execution
last_date	timestamp without time zone	Start time of the last job execution
last_suc	text	Start time of the last successful job execution
this_date	timestamp without time zone	Start time of an ongoing job execution
this_suc	text	Start time of an ongoing successful job execution
next_date	timestamp without time zone	Schedule time of the next job execution
next_suc	text	Schedule time of the next successful job execution
broken	text	The value is **y** if the job state is broken and **n** if otherwise.
status	"char"	Status of the current job. The value can be **r**, **s**, **f**, or **d**. The default value is **r**. Status of job step: ● r=running ● s=successfully finished ● f= job failed ● d=aborted
interval	text	Time expression used to calculate the next time the job will be executed. If this parameter is set to **null**, the job will be executed once only.
failures	smallint	Number of times the job has started and failed. If a job fails to be executed for 16 consecutive times, no more attempt will be made on it.
what	text	Executable job

13.3.112 MY_OBJECTS

MY_OBJECTS displays all database objects accessible to the current user. This view exists in both **PG_CATALOG** and **SYS schema**.

Table 13-218 MY_OBJECTS columns

Name	Type	Description
object_name	name	Object name
object_id	oid	Object OID
object_type	name	Object type (**TABLE**, **INDEX**, **SEQUENCE**, or **VIEW**)
namespace	oid	Namespace that an object belongs to
created	timestamp with time zone	Creation time of an object
last_ddl_time	timestamp with time zone	Last modification time of an object

NOTICE

For details about the value ranges of **created** and **last_ddl_time**, see **PG_OBJECT**.

13.3.113 MY_PART_INDEXES

MY_PART_INDEXES displays information about partitioned table indexes accessible to the current user. This view exists in both **PG_CATALOG** and **SYS schema**.

Table 13-219 MY_PART_INDEXES columns

Name	Type	Description
def_tablespace_name	name	Tablespace name of a partitioned table index
index_owner	character varying(64)	Owner name of a partitioned table index
index_name	character varying(64)	Name of a partitioned table index
partition_count	bigint	Number of index partitions of a partitioned table index
partitioning_key_count	integer	Number of partition keys of a partitioned table

Name	Type	Description
partitioning_type	text	Partitioning policy of a partitioned table **NOTE** For details about the supported partitioning policies of the current partitioned table, see **CREATE TABLE PARTITION**.
schema	character varying(64)	Schema of a partitioned table index
table_name	character varying(64)	Name of the partitioned table to which a partitioned table index belongs
subpartitioning_type	text	Partitioning policy of a level-2 partitioned table. If the partitioned table is a level-1 partitioned table, **NONE** is displayed. **NOTE** For details about the supported partitioning policies of the current level-2 partitioned table, see **CREATE TABLE SUBPARTITION**.
def_subpartition_count	integer	Default number of level-2 partitions to be created. The value is **1** for a level-2 partitioned table and **0** for a level-1 partitioned table.
subpartitioning_key_count	integer	Number of level-2 partition keys of the partitioned table.

13.3.114 MY_PART_TABLES

MY_PART_TABLES displays information about partitioned tables accessible to the current user. This view exists in both **PG_CATALOG** and **SYS schema**.

Table 13-220 MY_PART_TABLES columns

Name	Type	Description
table_owner	character varying(64)	Owner name of a partitioned table
table_name	character varying(64)	Name of a partitioned table

Name	Type	Description
partitioning_type	text	Partitioning policy of a partitioned table **NOTE** For details about the supported partitioning policies of the current partitioned table, see **CREATE TABLE PARTITION**.
partition_count	bigint	Number of partitions of a partitioned table
partitioning_key_count	integer	Number of partition keys of a partitioned table
def_tablespace_name	name	Tablespace name of a partitioned table
schema	character varying(64)	Schema of a partitioned table
subpartitioning_type	text	Partitioning policy of a level-2 partitioned table. If the partitioned table is a level-1 partitioned table, **NONE** is displayed. **NOTE** For details about the supported partitioning policies of the current level-2 partitioned table, see **CREATE TABLE SUBPARTITION**.
def_subpartition_count	integer	Default number of level-2 partitions to be created. The value is **1** for a level-2 partitioned table and **0** for a level-1 partitioned table.
subpartitioning_key_count	integer	Number of level-2 partition keys of the partitioned table.

13.3.115 MY_PROCEDURES

MY_PROCEDURES displays information about all stored procedures and functions in the current schema. This view exists in both **PG_CATALOG** and **SYS schema**.

Table 13-221 MY_PROCEDURES columns

Name	Type	Description
owner	character varying(64)	Owner of a stored procedure or function

Name	Type	Description
object_name	character varying(64)	Name of a stored procedure or function
argument_number	smallint	Number of input parameters in a stored procedure

13.3.116 MY_SEQUENCES

MY_SEQUENCES displays information about sequences in the current schema. This view exists in both **PG_CATALOG** and **SYS schema**.

Table 13-222 MY_SEQUENCES columns

Name	Type	Description
sequence_owner	character varying(64)	Owner of a sequence
sequence_name	character varying(64)	Name of a sequence

13.3.117 MY_SOURCE

MY_SOURCE displays information about stored procedures or functions in this schema and provides columns defined by the stored procedures or the functions. This view exists in both **PG_CATALOG** and **SYS schema**.

Table 13-223 MY_SOURCE columns

Name	Type	Description
owner	character varying(64)	Owner of a stored procedure or function
name	character varying(64)	Name of a stored procedure or function
text	text	Definition of a stored procedure or function

13.3.118 MY_SYNONYMS

MY_SYNONYMS displays synonyms accessible to the current user.

Table 13-224 MY_SYNONYMS columns

Name	Type	Description
schema_name	text	Name of the schema to which a synonym belongs
synonym_name	text	Name of a synonym
table_owner	text	Owner of an associated object Although the column is called **table_owner**, the associated object owned by the column is not necessarily a table. It can be any common database object, such as a view, stored procedure, or synonym.
table_name	text	Name of an associated object Although the column is called **table_name**, the associated object is not necessarily a table. It can be any common database object, such as a view, stored procedure, or synonym.
table_schema_name	text	Schema name of an associated object Although the column is called **table_schema_name**, the associated object is not necessarily a table. It can be any common database object, such as a view, stored procedure, or synonym.

13.3.119 MY_TAB_COLUMNS

MY_TAB_COLUMNS displays information about table columns accessible to the current user. This view exists in both **PG_CATALOG** and **SYS schema**.

Table 13-225 MY_TAB_COLUMNS columns

Name	Type	Description
owner	character varying(64)	Table owner
table_name	character varying(64)	Table name
column_name	character varying(64)	Column name
data_type	character varying(128)	Data type of a column
data_length	integer	Length of a column, in bytes

Name	Type	Description
data_precision	integer	Precision of a data type. This parameter is valid for the numeric data type and **NULL** for other data types.
data_scale	integer	Number of decimal places. This parameter is valid for the numeric data type and is set to **0** for other data types.
nullable	bpchar	Whether a column can be empty (**n** for the primary key constraint and non-null constraint)
column_id	integer	Sequence number of a column when a table is created
avg_col_len	numeric	Average length of a column, in bytes
char_length	numeric	Length of a column, in characters. This parameter is valid only for the varchar, nvarchar2, bpchar, and char types.
comments	text	Comments

13.3.120 MY_TAB_COMMENTS

MY_TAB_COMMENTS displays comments about all tables and views accessible to the current user. This view exists in both **PG_CATALOG** and **SYS schema**.

Table 13-226 MY_TAB_COMMENTS columns

Name	Type	Description
owner	character varying(64)	Owner of a table or view
table_name	character varying(64)	Name of a table or view
comments	text	Comments

13.3.121 MY_TAB_PARTITIONS

MY_TAB_PARTITIONS displays all table partitions accessible to the current user. Each table partition of a partitioned table accessible to the current user has one record in **USER_TAB_PARTITIONS**. This view exists in both **PG_CATALOG** and **SYS schema**.

Table 13-227 MY_TAB_PARTITIONS columns

Name	Type	Description
table_owner	character varying(64)	Owner name of a partitioned table
table_name	character varying(64)	Name of a partitioned table
partition_name	character varying(64)	Name of a partition
high_value	text	Limit of a partition. **NOTE** ● For range partitioning and interval partitioning, the upper limit of each partition is displayed. ● For list partitioning, the value list of each partition is displayed. ● For hash partitioning, the number of each partition is displayed.
tablespace_name	name	Tablespace name of a partition
schema	character varying(64)	Schema of a partitioned table
subpartition_count	bigint	Number of level-2 partitions.
high_value_length	integer	Character length of the limit of a partition.

13.3.122 MY_TAB_SUBPARTITIONS

MY_TAB_SUBPARTITIONS displays information about level-2 partitions owned by the current user. This view exists in the **PG_CATALOG** and **SYS** schemas.

Table 13-228 MY_TAB_SUBPARTITIONS columns

Name	Type	Description
table_owner	character varying(64)	Role name
table_name	character varying(64)	Relational table name
partition_name	character varying(64)	Partition name
subpartition_name	character varying(64)	Name of a level-2 partition

Name	Type	Description
high_value	text	Limit of a level-2 partition **NOTE** ● For range partitioning and interval partitioning, the upper limit of each partition is displayed. ● For list partitioning, the value list of each partition is displayed. ● For hash partitioning, the number of each partition is displayed.
tablespace_name	name	Tablespace name of a level-2 partitioned table
schema	character varying(64)	Name of a namespace
high_value_length	integer	Character length of the limit of a level-2 partition

13.3.123 MY_TABLES

MY_TABLES displays table information in the current schema. This view exists in both **PG_CATALOG** and **SYS schema**.

Table 13-229 MY_TABLES columns

Name	Type	Description
owner	character varying(64)	Table owner
table_name	character varying(64)	Table name
tablespace_name	character varying(64)	Tablespace name of a table
dropped	character varying	Whether the current record is deleted ● **yes**: It is deleted. ● **no**: It is not deleted.
num_rows	numeric	Estimated number of rows in a table
status	character varying(8)	Whether the current record is valid ● **valid**: It is valid.

Name	Type	Description
temporary	character(1)	Whether a table is a temporary table ● **y**: It is a temporary table. ● **n**: It is not a temporary table.

13.3.124 MY_TRIGGERS

MY_TRIGGERS displays information about triggers accessible to the current user. This view exists in the **PG_CATALOG** and **SYS** schemas.

Table 13-230 MY_TRIGGERS columns

Name	Type	Description
trigger_name	character varying(64)	Trigger name
table_name	character varying(64)	Relational table name
table_owner	character varying(64)	Role name
status	character varying(64)	● **O**: The trigger is enabled in origin or local mode. ● **D**: The trigger is disabled. ● **R**: The trigger is enabled in replica mode. ● **A**: The trigger is always enabled.

13.3.125 MY_VIEWS

MY_VIEWS displays information about all views in the current schema. This view exists in both **PG_CATALOG** and **SYS schema**.

Table 13-231 MY_VIEWS columns

Name	Type	Description
owner	character varying(64)	View owner
view_name	character varying(64)	View name

13.3.126 PG_AVAILABLE_EXTENSION_VERSIONS

PG_AVAILABLE_EXTENSION_VERSIONS displays extension versions of certain database features.

Table 13-232 PG_AVAILABLE_EXTENSION_VERSIONS columns

Name	Type	Description
name	name	Extension name
version	text	Version name
installed	boolean	Whether this extension version is installed
superuser	boolean	Whether only system administrators are allowed to install the extension
relocatable	boolean	Whether the extension can be relocated to another schema
schema	name	Name of the schema that the extension must be installed into (**NULL** if the extension is partially or fully relocatable)
requires	name[]	Names of prerequisite extensions (**NULL** if none)
comment	text	Comment string from the extension's control file

13.3.127 PG_AVAILABLE_EXTENSIONS

PG_AVAILABLE_EXTENSIONS displays extended information about certain database features.

Table 13-233 PG_AVAILABLE_EXTENSIONS columns

Name	Type	Description
name	name	Extension name
default_version	text	Name of the default version (**NULL** if none is specified)
installed_version	text	Currently installed version of the extension (**NULL** if no version is installed)
comment	text	Comment string from the extension's control file

13.3.128 PG_CURSORS

PG_CURSORS displays cursors that are currently available.

Table 13-234 PG_CURSORS columns

Name	Type	Description
name	text	Cursor name
statement	text	Query statement when the cursor is declared to change
is_holdable	boolean	**True** if the cursor is holdable (it can be accessed after the transaction that declared the cursor has committed); **false** otherwise
is_binary	boolean	Whether the cursor was declared BINARY. If it was, the value is **TRUE**.
is_scrollable	boolean	Whether the cursor is scrollable (it allows rows to be retrieved in a nonsequential manner). If it is, the value is **TRUE**.
creation_time	timestamp with time zone	Timestamp at which the cursor is declared

13.3.129 PG_COMM_DELAY

PG_COMM_DELAY displays the communication library delay status for a single DN.

Table 13-235 PG_COMM_DELAY columns

Name	Type	Description
node_name	text	Node name
remote_name	text	Name of the peer node
remote_host	text	IP address of the peer node
stream_num	integer	Number of logical stream connections used by the current physical connection
min_delay	integer	Minimum delay of the current physical connection within 1 minute, in microsecond **NOTE** A negative result is invalid. Wait until the delay status is updated and query again.
average	integer	Average delay of the current physical connection within 1 minute, in microsecond
max_delay	integer	Maximum delay of the current physical connection within 1 minute, in microsecond

13.3.130 PG_COMM_RECV_STREAM

PG_COMM_RECV_STREAM displays the receiving stream status of all the communication libraries for a single DN.

Table 13-236 PG_COMM_RECV_STREAM columns

Name	Type	Description
node_name	text	Node name
local_tid	bigint	ID of the thread using this stream
remote_name	text	Name of the peer node
remote_tid	bigint	Peer thread ID
idx	integer	Peer DN ID in the local DN
sid	integer	Stream ID in the physical connection
tcp_sock	integer	TCP socket used in the stream
state	text	Status of the stream ● **UNKNOWN**: The logical connection status is unknown. ● **READY**: The logical connection is ready. ● **RUN**: The logical connection sends packets normally. ● **HOLD**: The logical connection is waiting to send packets. ● **CLOSED**: The logical connection is closed. ● **TO_CLOSED**: The logical connection will be closed.
query_id	bigint	**debug_query_id** corresponding to the stream
pn_id	integer	**plan_node_id** of the query executed by the stream
send_smp	integer	**smpid** of the sender of the query executed by the stream
recv_smp	integer	**smpid** of the receiver of the query executed by the stream
recv_bytes	bigint	Total data volume received from the stream, in byte
time	bigint	Current lifecycle service duration of the stream, in ms
speed	bigint	Average receiving rate of the stream, in byte/s

Name	Type	Description
quota	bigint	Current communication quota value of the stream, in byte
buff_usize	bigint	Current size of the data cache of the stream, in byte

13.3.131 PG_COMM_SEND_STREAM

PG_COMM_SEND_STREAM displays the sending stream status of all the communication libraries for a single DN.

Table 13-237 PG_COMM_SEND_STREAM columns

Name	Type	Description
node_name	text	Node name
local_tid	bigint	ID of the thread using this stream
remote_name	text	Name of the peer node
remote_tid	bigint	Peer thread ID
idx	integer	Peer DN ID in the local DN
sid	integer	Stream ID in the physical connection
tcp_sock	integer	TCP socket used in the stream
state	text	Status of the stream ● **UNKNOWN**: The logical connection status is unknown. ● **READY**: The logical connection is ready. ● **RUN**: The logical connection sends packets normally. ● **HOLD**: The logical connection is waiting to send packets. ● **CLOSED**: The logical connection is closed. ● **TO_CLOSED**: The logical connection will be closed.
query_id	bigint	**debug_query_id** corresponding to the stream
pn_id	integer	**plan_node_id** of the query executed by the stream
send_smp	integer	**smpid** of the sender of the query executed by the stream

Name	Type	Description
recv_smp	integer	**smpid** of the receiver of the query executed by the stream
send_bytes	bigint	Total data volume sent by the stream, in byte
time	bigint	Current lifecycle service duration of the stream, in ms
speed	bigint	Average sending rate of the stream, in byte/s
quota	bigint	Current communication quota value of the stream, in byte
wait_quota	bigint	Extra time generated when the stream waits the quota value, in ms

13.3.132 PG_COMM_STATUS

PG_COMM_STATUS displays the communication library status for a single DN.

Table 13-238 PG_COMM_STATUS columns

Name	Type	Description
node_name	text	Node name
rxpck_rate	integer	Receiving rate of the communication library on the node, in byte/s
txpck_rate	integer	Sending rate of the communication library on the node, in byte/s
rxkbyte_rate	bigint	Receiving rate of the communication library on the node, in kbyte/s
txkbyte_rate	bigint	Sending rate of the communication library on the node, in kbyte/s
buffer	bigint	Size of the buffer of the Cmailbox
memkbyte_libcomm	bigint	Communication memory size of the **libcomm** process, in bytes
memkbyte_libpq	bigint	Communication memory size of the **libpq** process, in bytes
used_pm	integer	Real-time usage of the **postmaster** thread
used_sflow	integer	Real-time usage of the **gs_sender_flow_controller** thread

Name	Type	Description
used_rflow	integer	Real-time usage of the **gs_receiver_flow_controller** thread
used_rloop	integer	Highest real-time usage among multiple **gs_receivers_loop** threads
stream	integer	Total number of used logical connections.

13.3.133 PG_CONTROL_GROUP_CONFIG

PG_CONTROL_GROUP_CONFIG stores Cgroup configuration information in the system. Only the user with sysadmin permission can query this view.

Table 13-239 PG_CONTROL_GROUP_CONFIG columns

Name	Type	Description
pg_control_group_config	text	Configuration information of the Cgroup

13.3.134 PG_EXT_STATS

PG_EXT_STATS allows for access to extension statistics stored in the **PG_STATISTIC_EXT** system catalog. Currently, the extended statistics include the multi-column statistics. The current feature is a lab feature. Contact Huawei technical support before using it.

Table 13-240 PG_EXT_STATS columns

Name	Type	Reference	Description
schemaname	name	**PG_NAMESPACE**.nspname	Name of the schema that contains a table
tablename	name	**PG_CLASS**.relname	Table name
attname	int2vector	**PG_STATISTIC_EXT**.stakey	Columns to be combined for collecting statistics
inherited	boolean	-	Includes inherited sub-columns if the value is **true**; otherwise, it indicates the column in a specified table.
null_frac	real	-	Percentage of column combinations that are null to all records

Name	Type	Reference	Description
avg_width	integer	-	Average width of column combinations, in byte
n_distinct	real	-	● Estimated number of distinct values in a column combination if the value is greater than 0 ● Negative of the number of distinct values divided by the number of rows if the value is less than 0 1. The negated form is used when **ANALYZE** believes that the number of distinct values is likely to increase as the table grows. 2. The positive form is used when the column seems to have a fixed number of possible values. For example, **–1** indicates that the number of distinct values is the same as the number of rows for a column combination. ● The number of distinct values is unknown if the value is **0**.
n_dndistinct	real	-	Number of unique not-null data values in the **dn1** column combination ● Exact number of distinct values if the value is greater than **0**. ● Negative of the number of distinct values divided by the number of rows if the value is less than 0 For example, if a value in a column combination appears twice in average, **n_dndistinct** equals **–0.5**. ● The number of distinct values is unknown if the value is **0**.

Name	Type	Reference	Description
most_common_vals	anyarray	-	List of the most common values in a column combination. If this combination does not have the most common values, **most_common_vals** will be **NULL**. None of the most common values in **most_common_vals** is **NULL**.
most_common_freqs	real[]	-	List of the frequencies of the most common values, that is, the number of occurrences of each value divided by the total number of rows (**NULL** if **most_common_vals** is **NULL**)
most_common_vals_null	anyarray	-	List of the most common values in a column combination. If this combination does not have the most common values, **most_common_vals** will be **NULL**. At least one of the common values in **most_common_vals_null** is **NULL**.
most_common_freqs_null	real[]	-	List of the frequencies of the most common values, that is, the number of occurrences of each value divided by the total number of rows (**NULL** if **most_common_vals_null** is **NULL**)
histogram_bounds	anyarray	-	Boundary value list of the histogram

13.3.135 PG_GET_INVALID_BACKENDS

PG_GET_INVALID_BACKENDS provides information about background threads on the primary database node that are connected to the current standby server.

Table 13-241 PG_GET_INVALID_BACKENDS columns

Name	Type	Description
pid	bigint	Thread ID
node_name	text	Node information connected to the background thread

Name	Type	Description
dbname	name	Name of the connected database
backend_start	timestamp with time zone	Background thread startup time
query	text	Query statement executed by the background thread

13.3.136 PG_GET_SENDERS_CATCHUP_TIME

PG_GET_SENDERS_CATCHUP_TIME provides catchup information of the currently active primary/standby instance sender thread on the database node.

Table 13-242 PG_GET_SENDERS_CATCHUP_TIME columns

Name	Type	Description
pid	bigint	Current sender thread ID
lwpid	integer	Current sender lwpid
local_role	text	Local role
peer_role	text	Peer role
state	text	Current sender's replication status
type	text	Current sender type
catchup_start	timestamp with time zone	Startup time of a catchup task
catchup_end	timestamp with time zone	End time of a catchup task

13.3.137 PG_GROUP

PG_GROUP displays the database role authentication and the relationship between roles.

Table 13-243 PG_GROUP columns

Name	Type	Description
groname	name	Group name

Name	Type	Description
grosysid	oid	Group ID
grolist	oid[]	An array, including all the role IDs in this group

13.3.138 PG_GTT_RELSTATS

PG_GTT_RELSTATS displays basic information about all global temporary tables of the current session by invoking **pg_get_gtt_relstats**.

Table 13-244 PG_GTT_RELSTATS columns

Name	Type	Description
schemaname	name	Schema name
tablename	name	Name of a global temporary table
relfilenode	oid	ID of a file object
relpages	integer	Number of disk pages of a global temporary table
reltuples	real	Number of records in a global temporary table
relallvisible	integer	Number of pages that are marked as all visible
relfrozenxid	xid	All transaction IDs before this one have been replaced with a permanent (frozen) transaction ID in the table.
relminmxid	xid	Reserved

13.3.139 PG_GTT_STATS

PG_GTT_STATS displays statistics about a single column in all global temporary tables of the current session by calling **pg_get_gtt_relstats**.

Table 13-245 PG_GTT_STATS columns

Name	Type	Description
schemaname	name	Schema name.
tablename	name	Name of a global temporary table.
attname	name	Attribute name.

Name	Type	Description
inherited	boolean	Specifies whether to collect statistics for objects that have inheritance relationship.
null_frac	real	Percentage of column entries that are null.
avg_width	integer	Average stored width, in bytes, of non-null entries.
n_distinct	real	Number of distinct, non-null data values in the column.
most_common_vals	text[]	List of the most common values, which is sorted by occurrence frequency.
most_common_freqs	real[]	Frequencies of the most common values.
histogram_bounds	text[]	Data distribution (excluding the most common values) in a frequency histogram description column.
correlation	real	Correlation coefficient.
most_common_elems	text[]	List of the most common element values, which is used for the array type or some other type.
most_common_elem_freqs	real[]	Frequencies of the most common element values.
elem_count_histogram	real[]	Array type histogram.

13.3.140 PG_GTT_ATTACHED_PIDS

PG_GTT_ATTACHED_PIDS checks which sessions are using global temporary tables by calling the **pg_get_attached_pid** function.

Table 13-246 PG_GTT_ATTACHED_PIDS columns

Name	Type	Description
schemaname	name	Schema name
tablename	name	Name of a global temporary table
relid	oid	OID of a global temporary table
pids	bigint[]	Thread PID list
sessionids	bigint[]	Session ID list

13.3.141 PG_INDEXES

PG_INDEXES provides access to useful information about each index in the database.

Table 13-247 PG_INDEXES columns

Name	Type	Reference	Description
schemaname	name	**PG_NAMESPACE**.nspname	Name of the schema that contains tables and indexes
tablename	name	**PG_CLASS**.relname	Name of the table for which the index serves
indexname	name	**PG_CLASS**.relname	Index name
tablespace	name	**PG_TABLESPACE**.nspname	Name of the tablespace that contains the index
indexdef	text	-	Index definition (a reconstructed **CREATE INDEX** command)

13.3.142 PG_LOCKS

PG_LOCKS displays information about locks held by open transactions.

Table 13-248 PG_LOCKS columns

Name	Type	Reference	Description
locktype	text	-	Type of the locked object: relation, extend, page, tuple, transactionid, virtualxid, object, userlock, and advisory
database	oid	OID in **PG_DATABASE**	OID of the database in which the locked target exists ● The OID is **0** if the target is a shared object. ● The OID is **NULL** if the locked target is a transaction.
relation	oid	OID in **PG_CLASS**	OID of the relationship targeted by the lock (**NULL** if the object is not a relation or part of a relation)

Name	Type	Reference	Description
page	integer	-	Page number targeted by the lock within the relation (**NULL** if the object is not a relation page or row page)
tuple	smallint	-	Row number targeted by the lock within the page (**NULL** if the object is not a row)
bucket	integer	-	Bucket number corresponding to the child table. The value is **NULL** if the target is not a table.
virtualxid	text	-	Virtual ID of the transaction targeted by the lock (**NULL** if the object is not a virtual transaction)
transactionid	xid	-	ID of the transaction targeted by the lock (**NULL** if the object is not a transaction)
classid	oid	OID in **PG_CLASS**	OID of the system catalog that contains the object (**NULL** if the object is not a general database object)
objid	oid	-	OID of the lock target within its system table (**NULL** if the target is not a general database object)
objsubid	smallint	-	Column number for a column in the table (**0** if the object is of other object type and **NULL** if the object is not a general database object)
virtualtransaction	text	-	Virtual ID of the transaction holding or awaiting this lock
pid	bigint	-	Logical ID of the server thread holding or awaiting this lock (**NULL** if the lock is held by a prepared transaction)
sessionid	bigint	-	ID of the session that holds or waits for the lock
mode	text	-	Lock mode held or desired by this thread
granted	boolean	-	● The value is **TRUE** if the lock is a held lock. ● The value is **FALSE** if the lock is an awaited lock.

Name	Type	Reference	Description
fastpath	boolean	-	The value is **TRUE** if the lock is obtained through **fast-path**, and is **FALSE** if the lock is obtained through the main lock table.
locktag	text	-	Lock information that the session waits for. It can be parsed using the **locktag_decode()** function.
global_sessionid	text	-	Global session ID

13.3.143 PG_NODE_ENV

PG_NODE_ENV obtains environmental variable information about the current node.

Table 13-249 PG_NODE_ENV columns

Name	Type	Description
node_name	text	Current node name
host	text	Host name of the node
process	integer	Number of the node process
port	integer	Port ID of the node
installpath	text	Installation directory of the node
datapath	text	Data directory of the node
log_directory	text	Log directory of the node

13.3.144 PG_OS_THREADS

PG_OS_THREADS provides status information about all the threads under the current node.

Table 13-250 PG_OS_THREADS columns

Name	Type	Description
node_name	text	Current node name
pid	bigint	PID of the thread running under the current node process

Name	Type	Description
lwpid	integer	Lightweight thread ID corresponding to the PID
thread_name	text	Thread name corresponding to the PID
creation_time	timestamp with time zone	Thread creation time corresponding to the PID

13.3.145 PG_PREPARED_STATEMENTS

PG_PREPARED_STATEMENTS displays all prepared statements that are available in the current session.

Table 13-251 PG_PREPARED_STATEMENTS columns

Name	Type	Description
name	text	Identifier of the prepared statement
statement	text	Query string for creating this prepared statement. For prepared statements created through SQL, this is the PREPARE statement submitted by the client. For prepared statements created through the frontend/ backend protocol, this is the text of the prepared statement itself.
prepare_time	timestamp with time zone	Timestamp when the prepared statement is created
parameter_types	regtype[]	Expected parameter types for the prepared statement in the form of an array of **regtype**. The OID corresponding to an element of this array can be obtained by converting the **regtype** value to **oid**.
from_sql	boolean	● **True** if the prepared statement was created through the PREPARE statement ● **False** if the statement was prepared through the frontend/backend protocol

13.3.146 PG_PREPARED_XACTS

PG_PREPARED_XACTS displays information about transactions that are currently prepared for two-phase commit.

Table 13-252 PG_PREPARED_XACTS columns

Name	Type	Reference	Description
transaction	xid	-	Numeric transaction identifier of the prepared transaction
gid	text	-	Global transaction identifier that was assigned to the transaction
prepared	timestamp with time zone	-	Time at which the transaction is prepared for commit
owner	name	**PG_AUTHID**.rolname	Name of the user that executes the transaction
database	name	**PG_DATABASE**.datname	Name of the database in which the transaction is executed

13.3.147 PG_PUBLICATION_TABLES

PG_PUBLICATION_TABLES displays the mapping information between a publication and its published tables. Unlike the underlying system catalog **PG_PUBLICATION_REL**, this view expands publications defined as **FOR ALL TABLES** so that for such publications, there is one row for each eligible table.

Table 13-253 PG_PUBLICATION_TABLES columns

Name	Type	Description
pubname	name	Publication name.
schemaname	name	Name of the schema that contains the table.
tablename	name	Table name.

13.3.148 PG_REPLICATION_ORIGIN_STATUS

PG_REPLICATION_ORIGIN_STATUS displays the replication status of the replication source.

Table 13-254 PG_REPLICATION_ORIGIN_STATUS columns

Name	Type	Description
local_id	oid	Replication source ID.

Name	Type	Description
external_id	text	Name of the replication source.
remote_lsn	text	Remote LSN of the replication source.
local_lsn	text	Local LSN of the replication source.

13.3.149 PG_REPLICATION_SLOTS

PG_REPLICATION_SLOTS contains replication slot information.

Table 13-255 PG_REPLICATION_SLOTS columns

Name	Type	Description
slot_name	text	Replication slot name.
plugin	text	Name of the output plug-in corresponding to the logical replication slot.
slot_type	text	Replication slot type. ● **physical**: physical replication slot. ● **logical**: logical replication slot.
datoid	oid	OID of the database where the replication slot resides.
database	name	Name of the database where the replication slot resides.
active	boolean	Determines whether the replication slot is activated. ● **t** (true): yes ● **f** (false): no
xmin	xid	XID of the earliest transaction that the database must reserve for the replication slot.
catalog_xmin	xid	XID of the earliest system catalog-involved transaction that the database must reserve for the logical replication slot.
restart_lsn	text	Physical location of the earliest Xlog required by the replication slot.

Name	Type	Description
dummy_standby	boolean	Determines whether the peer end connected to the replication slot is a secondary node. ● **t** (true): yes ● **f** (false): no
confirmed_flush	text	Dedicated logical replication slot. The client confirms the location of the received log.

13.3.150 PG_RLSPOLICIES

PG_RLSPOLICIES contains row-level access control policies.

Table 13-256 PG_RLSPOLICIES columns

Name	Type	Description
schemaname	name	Name of the schema of the table object to which a row-level access control policy is applied
tablename	name	Name of the table object to which the row-level access control policy is applied
policyname	name	Name of the row-level access control policy
policypermissive	text	Attribute of the row-level access control policy
policyroles	name[]	List of users affected by the row-level access control policy. If this parameter is not specified, all users are affected.
policycmd	text	SQL operations affected by the row-level access control policy
policyqual	text	Expression of the row-level access control policy

13.3.151 PG_ROLES

PG_ROLES provides information about database roles. Initialization users and users with the sysadmin or createrole attribute can view information about all roles. Other users can view only their own information.

Table 13-257 PG_ROLES columns

Name	Type	Reference	Description
rolname	name	-	Role name

Name	Type	Reference	Description
rolsuper	boolean	-	Whether the role is the initial system administrator with the highest permission
rolinherit	boolean	-	Whether the role inherits the permissions for this type of roles
rolcreaterole	boolean	-	Whether the role can create other roles
rolcreatedb	boolean	-	Whether the role can create databases
rolcatupdate	boolean	-	Whether the role can update system tables directly. Only the initial system administrator whose **usesysid** is **10** has this permission. It is unavailable for other users.
rolcanlogin	boolean	-	Whether the role can log in to the database
rolreplication	boolean	-	Whether the role can be replicated
rolauditadmin	boolean	-	Whether the role is an audit system administrator
rolsystemadmin	boolean	-	Whether the role is a system administrator
rolconnlimit	integer	-	Sets the maximum number of concurrent connections that this role can initiate if this role can log in. The value **–1** indicates no limit.
rolpassword	text	-	Not the password (always reads as ********)
rolvalidbegin	timestamp with time zone	-	Account validity period start time; null if no start time
rolvaliduntil	timestamp with time zone	-	Account expiry time; null if no expiration
rolrespool	name	-	Resource pool that a user can use
rolparentid	oid	**PG_AUTHID**.rolparentid	OID of a group user to which the user belongs
roltabspace	text	-	Storage space of the user permanent table

Name	Type	Reference	Description
roltempspace	text	-	Storage space of the user temporary table, in KB
rolspillspace	text	-	Operator disk spill space of the user, in KB
rolconfig	text[]	-	Session defaults for runtime configuration variables
oid	oid	**PG_AUTHID**.oid	Role ID
roluseft	boolean	**PG_AUTHID**.roluseft	Whether the role can perform operations on foreign tables
rolkind	"char"	-	Role type
nodegroup	name	-	Unsupported currently
rolmonitoradmin	boolean	-	Whether the role is a monitor administrator
roloperatoradmin	boolean	-	Whether the role is an O&M administrator
rolpolicyadmin	boolean	-	Whether the role is a security policy administrator

13.3.152 PG_RULES

PG_RULES provides access to query useful information about rewrite rules.

Table 13-258 PG_RULES columns

Name	Type	Description
schemaname	name	Name of the schema that contains a table
tablename	name	Name of the table to which the rule applies
rulename	name	Name of a rule
definition	text	Rule definition (a reconstructed creation command)

13.3.153 PG_RUNNING_XACTS

PG_RUNNING_XACTS displays the running transaction information on the current node.

Table 13-259 PG_RUNNING_XACTS columns

Name	Type	Description
handle	integer	Slot handle in the transaction manager corresponding to a transaction. The value is fixed at **–1**.
gxid	xid	Transaction ID
state	tinyint	Transaction status (**3**: prepared or **0**: starting)
node	text	Node name
xmin	xid	Minimum transaction ID on the node
vacuum	boolean	Whether the current transaction is lazy vacuum ● **t** (true): yes ● **f** (false): no
timeline	bigint	Number of database restarts
prepare_xid	xid	Transaction ID in the **prepared** state. If the status is not **prepared**, the value is **0**.
pid	bigint	Thread ID corresponding to the transaction
next_xid	xid	Transaction ID sent by other nodes to the current node. The value is fixed at **0**.

13.3.154 PG_SECLABELS

PG_SECLABELS provides information about security labels.

Table 13-260 PG_SECLABELS columns

Name	Type	Reference	Description
objoid	oid	Any OID column	OID of the object that this security label pertains to
classoid	oid	OID in **PG_CLASS**	OID of the system catalog where the object appears
objsubid	intege r	-	Column number for the security label on a table column (**objoid** and **classoid** refer to the table itself). The value is **0** for all other object types.
objtype	text	-	Type of object to which this label applies, as text

Name	Type	Reference	Description
objnamespace	oid	OID in **PG_NAMESPACE**	OID of the namespace for this object, if applicable; otherwise **NULL**
objname	text	-	Text-typed name of the object to which this label applies
provider	text	Provider in **PG_SECLABEL**	Label provider associated with the label
label	text	Label in **PG_SECLABEL**	Security label applied to the object

13.3.155 PG_SESSION_IOSTAT

PG_SESSION_IOSTAT shows I/O load management information about the task currently executed by the user. The current feature is a lab feature. Contact Huawei technical support before using it. Only users with the **sysadmin** or **monitor admin** permission can query this view.

IOPS is counted by ones for column store and by 10 thousands for row store.

Table 13-261 PG_SESSION_IOSTAT columns

Name	Type	Description
query_id	bigint	Job ID
mincurriops	integer	Minimum I/O of the job across database instances
maxcurriops	integer	Maximum I/O of the job across database instances
minpeakiops	integer	Minimum peak I/O of the current job across database instances
maxpeakiops	integer	Maximum peak I/O of the current job across database instances
io_limits	integer	**io_limits** set for the job
io_priority	text	**io_priority** set for the job
query	text	Job
node_group	text	Unsupported currently
curr_io_limits	integer	Real-time **io_limits** value when **io_priority** is used to control I/Os

13.3.156 PG_SESSION_WLMSTAT

PG_SESSION_WLMSTAT displays corresponding load management information about the task currently executed by the user. The current feature is a lab feature. Contact Huawei technical support before using it. Only the user with sysadmin permission can query this view.

Table 13-262 PG_SESSION_WLMSTAT columns

Name	Type	Description
datid	oid	OID of the database this backend is connected to.
datname	name	Name of the database the backend is connected to.
threadid	bigint	ID of the backend thread.
sessionid	bigint	Session ID.
processid	integer	PID of the backend thread.
usesysid	oid	OID of the user logged in to the backend.
appname	text	Name of the application connected to the backend.
usename	name	Name of the user logged in to the backend.
priority	bigint	Priority of Cgroup where the statement is located.
attribute	text	Attributes of the statement: ● **Ordinary**: default attribute of a statement before it is parsed by the database. ● **Simple**: simple statement. ● **Complicated**: complicated statements. ● **Internal**: internal statement of the database. ● **Unknown**: unknown.
block_time	bigint	Pending duration of the statement by now, in seconds.
elapsed_time	bigint	Actual execution duration of the statement by now, in seconds.
total_cpu_time	bigint	Total CPU usage duration of the statement on the database instances in the last period (unit: s).
cpu_skew_percent	integer	CPU usage skew percentage of the statement on the database instances in the last period.
statement_mem	integer	**statement_mem** used for executing the statement (reserved column).

Name	Type	Description
active_points	integer	Number of concurrently active points occupied by the statement in the resource pool.
dop_value	integer	DOP value obtained by the statement from the resource pool.
control_group	text	Unsupported currently.
status	text	Specifies the status of the statement, including: ● **pending**: waiting to be executed. ● **running**: being executed. ● **finished**: finished normally. (If **enqueue** is set to **StoredProc** or **Transaction**, this state indicates that only certain jobs in the statement have been executed. This state persists until the finish of this statement.) ● **aborted**: terminated unexpectedly. ● **active**: normal status except for those above. ● **unknown**: unknown status.
enqueue	text	Unsupported currently.
resource_pool	name	Specifies the current resource pool where the statements are located.
query	text	Specifies the latest query at the backend. If **state** is **active**, this column shows the executing query. In all other states, it shows the last query that was executed.
is_plana	boolean	Unsupported currently.
node_group	text	Unsupported currently.

13.3.157 PG_SETTINGS

PG_SETTINGS displays information about parameters of the running database.

Table 13-263 PG_SETTINGS columns

Name	Type	Description
name	text	Parameter name
setting	text	Current parameter value
unit	text	Implicit unit of the parameter
category	text	Logical group of the parameter

Name	Type	Description
short_desc	text	Brief description of the parameter
extra_desc	text	Detailed description of the parameter
context	text	Context required to set the parameter value, including **internal**, **postmaster**, **sighup**, **backend**, **superuser**, and **user**
vartype	text	Parameter type, including **bool**, **enum**, **integer**, **real**, or **string**
source	text	Method of assigning the parameter value
min_val	text	Minimum value of the parameter. If the parameter type is not numeric, the value of this column is **null**.
max_val	text	Maximum value of the parameter. If the parameter type is not numeric, the value of this column is **null**.
enumvals	text[]	Valid values of an enum-type parameter. If the parameter type is not enum, the value of this column is **null**.
boot_val	text	Default parameter value used upon the database startup
reset_val	text	Default parameter value used upon the database reset
sourcefile	text	Configuration file used to set parameter values. If parameter values are not configured using the configuration file, the value of this column is **null**.
sourceline	integer	Row number of the configuration file for setting parameter values. If parameter values are not configured using the configuration file, the value of this column is **null**.

13.3.158 PG_SHADOW

PG_SHADOW displays properties of all roles that are marked as **rolcanlogin** in **PG_AUTHID**.

The name stems from the fact that this view should not be readable by the public since it contains passwords. **PG_USER** is a publicly readable view on **PG_SHADOW** that blanks out the password column.

Table 13-264 PG_SHADOW columns

Name	Type	Reference	Description
usename	name	**PG_AUTHID**.rolname	Indicates the username.
usesysid	oid	**PG_AUTHID**.oid	Indicates the user ID
usecreatedb	boolean	-	Indicates that the user can create databases.
usesuper	boolean	-	Indicates that the user is an administrator.
usecatupd	boolean	-	Indicates that the user can update views. Even the system administrator cannot do this unless this column is **true**.
userepl	boolean	-	Indicates that the user can initiate streaming replication and put the system in and out of backup mode.
passwd	text	-	Indicates the password (possibly encrypted); **null** if none. See **PG_AUTHID** for details about how encrypted passwords are stored.
valbegin	timestamp with time zone	-	Indicates the start time for account validity (**null** if no start time).
valuntil	timestamp with time zone	-	Indicates the end time for account validity (**null** if no end time)
respool	name	-	Indicates the resource pool used by the user.
parent	oid	-	Indicates the parent resource pool.
spacelimit	text	-	Indicates the storage space of the permanent table.
tempspacelimit	text	-	Indicates the storage space of the temporary table.
spillspacelimit	text	-	Indicates the operator disk flushing space.
useconfig	text[ ]	-	Indicates the session defaults for runtime configuration variables.

Name	Type	Reference	Description
usemonitor admin	boolean	-	Indicates whether the user is a monitor administrator. ● **t** (true): yes ● **f** (false): no
useoperator admin	boolean	-	Indicates whether the user is an O&M administrator. ● **t** (true): yes ● **f** (false): no
usepolicyad min	boolean	-	Indicates whether the user is a security policy administrator. ● **t** (true): yes ● **f** (false): no

13.3.159 PG_STATS

PG_STATS provides access to the single-column statistics stored in the **pg_statistic** table. The **autovacuum_naptime** parameter specifies the interval for updating statistics recorded in the view.

Table 13-265 PG_STATS columns

Name	Type	Reference	Description
schemanam e	name	**PG_NAMESPACE** .nspname	Name of the schema that contains a table
tablename	name	**PG_CLASS**.relna me	Table name
attname	name	**PG_ATTRIBUTE**.a ttname	Column name
inherited	boolean	-	Includes inherited sub-columns if the value is **true**; otherwise, it indicates the column in a specified table.
null_frac	real	-	Percentage of column entries that are null
avg_width	integer	-	Average width in bytes of column's entries

Name	Type	Reference	Description
n_distinct	real	-	● Estimated number of distinct values in the column if the value is greater than 0 ● Negative of the number of distinct values divided by the number of rows if the value is less than 0 1. The negated form is used when **ANALYZE** believes that the number of distinct values is likely to increase as the table grows. 2. The positive form is used when the column seems to have a fixed number of possible values. For example, value **–1** indicates that the number of distinct values is the same as the number of rows for a unique column.
n_dndistinct	real	-	Number of unique non-null data values in the **dn1** column ● Exact number of distinct values if the value is greater than **0**. ● Negative of the number of distinct values divided by the number of rows if the value is less than 0 For example, if the value of a column appears twice in average, set **n_dndistinct=-0.5**. ● The number of distinct values is unknown if the value is **0**.
most_common_vals	anyarray	-	List of the most common values in a column. (**NULL** if no values in the column seem to be more common than any others)
most_common_freqs	real[]	-	List of the frequencies of the most common values, that is, number of occurrences of each divided by total number of rows. (**NULL** if **most_common_vals** is **NULL**)

Name	Type	Reference	Description
histogram_bounds	anyarray	-	List of values that divide the column's values into groups of equal proportion. The values in **most_common_vals**, if present, are omitted from this histogram calculation. This column is null if the column data type does not have a **<** operator or if the **most_common_vals** list accounts for the entire population.
correlation	real	-	Statistical correlation between physical row ordering and logical ordering of the column values. It ranges from -1 to +1. When the value is near to -1 or +1, an index scan on the column is estimated to be cheaper than when it is near to zero, due to reduction of random access to the disk. This column is null if the column data type does not have a < operator.
most_common_elems	anyarray	-	A list of non-null element values most often appearing
most_common_elem_freqs	real[]	-	A list of the frequencies of the most common element values
elem_count_histogram	real[]	-	A histogram of the counts of distinct non-null element values

13.3.160 PG_STAT_ACTIVITY

PG_STAT_ACTIVITY shows information about the current user's queries. The columns save information about the last query.

Table 13-266 PG_STAT_ACTIVITY columns

Name	Type	Description
datid	oid	OID of the database that the user session connects to in the backend
datname	name	Name of the database that the user session connects to in the backend

Name	Type	Description
pid	bigint	ID of the backend thread
sessionid	bigint	Session ID
usesysid	oid	OID of the user logged in to the backend
usename	name	Name of the user logged in to the backend
application_name	text	Name of the application connected to the backend
client_addr	inet	IP address of the client connected to the backend. If this column is null, it indicates either the client is connected via a Unix socket on the server or this is an internal process, such as autovacuum.
client_hostname	text	Host name of the connected client, as reported by a reverse DNS lookup of **client_addr**. This column will be non-null only for IP connections and only when **log_hostname** is enabled.
client_port	integer	TCP port number that the client uses for communication with the backend (**–1** if a Unix socket is used)
backend_start	timestamp with time zone	Time when this process was started, that is, when the client connected to the server
xact_start	timestamp with time zone	Time when current transaction was started (null if no transaction is active) If the current query is the first of its transaction, the value of this column is the same as that of the **query_start** column.
query_start	timestamp with time zone	Time when the currently active query was started, or time when the last query was started if the value of **state** is not **active**
state_change	timestamp with time zone	Time when **state** was last modified
waiting	boolean	Whether the backend is currently waiting for a lock. If yes, the value is **true**.

Name	Type	Description
enqueue	text	Unsupported currently

state	text	Overall status of the backend. The value must be one of the following:
		● **active**: The backend is executing a query.
		● **idle**: The backend is waiting for a new client command.
		● **idle in transaction**: The backend is in a transaction, but there is no statement being executed in the transaction.
		● **idle in transaction** (aborted): The backend is in a transaction, but there are statements failed in the transaction.
		● **fastpath function call**: The backend is executing a fast-path function.
		● **disabled**: This state is reported if **track_activities** is disabled in this backend.
		NOTE
		Common users can view their own session status only. The state information of other accounts is empty. For example, after user **judy** is connected to the database, the state information of user **joe** and the initial user omm in **pg_stat_activity** is empty.
		`SELECT datname, usename, usesysid, state,pid FROM pg_stat_activity;`
		` datname  | usename | usesysid | state  | pid`
		`----------+---------+----------+--------+-----------------`
		` postgres | omm     |       10 |        | 139968752121616`
		` postgres | omm     |       10 |        | 139968903116560`
		` db_tpcc | judy    |    16398 | active | 139968391403280`
		` postgres | omm     |       10 |        | 139968643069712`
		` postgres | omm     |       10 |        | 139968680818448`
		` postgres | joe     |    16390 |        | 139968563377936`
		`(6 rows)`

resource_pool	name	Resource pool used by the user
query_id	bigint	ID of a query

Name	Type	Description
query	text	Latest query at the backend. If the value of **state** is **active**, this column shows the ongoing query. In all other states, it shows the last query that was executed.
connection_info	text	A string in JSON format recording the driver type, driver version, driver deployment path, and process owner of the connected database. For details, see **connection_info**.
unique_sql_id	bigint	Unique SQL statement ID.
trace_id	text	Driver-specific trace ID, which is associated with an application request.

13.3.161 PG_STAT_ACTIVITY_NG

PG_STAT_ACTIVITY_NG displays information about all queries in the logical database instance of the current user.

Table 13-267 PG_STAT_ACTIVITY_NG columns

Name	Type	Description
datid	oid	OID of the database that the user session connects to in the backend
datname	name	Name of the database that the user session connects to in the backend
pid	bigint	ID of the backend thread
sessionid	bigint	Session ID
usesysid	oid	OID of the user logged in to the backend
usename	name	Name of the user logged in to the backend
application_name	text	Name of the application connected to the backend

Name	Type	Description
client_addr	inet	IP address of the client connected to the backend If this column is null, it indicates either that the client is connected via a Unix socket on the server machine or this is an internal process such as autovacuum.
client_hostname	text	Host name of the connected client, as reported by a reverse DNS lookup of **client_addr**. This column will be non-null only for IP connections and only when **log_hostname** is enabled.
client_port	integer	TCP port number that the client uses for communication with the backend (**–1** if a Unix socket is used)
backend_start	timestamp with time zone	Time when this process was started, that is, when the client connected to the server
xact_start	timestamp with time zone	Time when current transaction was started (null if no transaction is active). If the current query is the first of its transaction, this column is equal to the **query_start** column.
query_start	timestamp with time zone	Time when the currently active query was started, or time when the last query was started if the value of **state** is not **active**
state_change	timestamp with time zone	Time when **state** was last modified
waiting	boolean	Whether the backend is currently waiting for a lock. If yes, the value is **true**. Otherwise, the value is **false**.
enqueue	text	Queuing status of a statement. Possible values are: ● **waiting in queue**: The statement is in the queue. ● **Empty**: The statement is running.

Name	Type	Description

state	text	Overall status of the backend. Its value can be:
		● **active**: The backend is executing a query.
		● **idle**: The backend is waiting for a new client command.
		● **idle in transaction**: The backend is in a transaction, but there is no statement being executed in the transaction.
		● **idle in transaction** (aborted): The backend is in a transaction, but there are statements failed in the transaction.
		● **fastpath function call**: The backend is executing a fast-path function.
		● **disabled**: This state is reported if **track_activities** is disabled in this backend.
		NOTE
		Common users can view status of their own sessions only. The state information of other accounts is empty. For example, after user **judy** is connected to the database, the state information of user **joe** and the initial user **omm** in **pg_stat_activity** is empty.
		`SELECT datname, usename, usesysid, state,pid FROM pg_stat_activity_ng;`
		` datname  | usename | usesysid | state  | pid`
		`----------+---------+----------+--------+-----------------`
		` postgres | omm     |       10 |        | 139968752121616`
		` postgres | omm     |       10 |        | 139968903116560`
		` db_tpcds | judy    |    16398 | active | 139968391403280`
		` postgres | omm     |       10 |        | 139968643069712`
		` postgres | omm     |       10 |        | 139968680818448`
		` postgres | joe     |    16390 |        | 139968563377936`
		`(6 rows)`

resource_pool	name	Resource pool used by the user
query_id	bigint	Query ID

Name	Type	Description
query	text	Latest query at the backend. If the value of **state** is **active**, this column shows the ongoing query. In all other states, it shows the last query that was executed.
node_group	text	Logical database instance of the user to which the statement belongs.

13.3.162 PG_STAT_ALL_INDEXES

PG_STAT_ALL_INDEXES contains one row for each index in the current database, showing statistics about accesses to that specific index.

Indexes can be used via either simple index scans or "bitmap" index scans. In a bitmap scan the output of several indexes can be combined via AND or OR rules, so it is difficult to associate individual heap row fetches with specific indexes when a bitmap scan is used. Therefore, a bitmap scan increments the **pg_stat_all_indexes.idx_tup_read** count(s) for the index(es) it uses, and it increments the **pg_stat_all_tables.idx_tup_fetch** count for the table, but it does not affect **pg_stat_all_indexes.idx_tup_fetch**.

Table 13-268 PG_STAT_ALL_INDEXES columns

Name	Type	Description
relid	oid	OID of the table for this index
indexrelid	oid	OID of this index
schemaname	name	Name of the schema that the index is in
relname	name	Name of the table for the index
indexrelname	name	Index name
idx_scan	bigint	Number of index scans initiated on the index
idx_tup_read	bigint	Number of index entries returned by scans on the index
idx_tup_fetch	bigint	Number of live table rows fetched by simple index scans using the index

13.3.163 PG_STAT_ALL_TABLES

PG_STAT_ALL_TABLES contains one row for each table in the current database (including TOAST tables), showing statistics about accesses to that specific table.

Table 13-269 PG_STAT_ALL_TABLES columns

Name	Type	Description
relid	oid	OID of the table
schemaname	name	Name of the schema that the table is in
relname	name	Table name
seq_scan	bigint	Number of sequential scans initiated on the table
seq_tup_read	bigint	Number of live rows fetched by sequential scans
idx_scan	bigint	Number of index scans initiated on the table
idx_tup_fetch	bigint	Number of live rows fetched by index scans
n_tup_ins	bigint	Number of rows inserted
n_tup_upd	bigint	Number of rows updated
n_tup_del	bigint	Number of rows deleted
n_tup_hot_upd	bigint	Number of rows HOT updated (that is, with no separate index update required)
n_live_tup	bigint	Estimated number of live rows
n_dead_tup	bigint	Estimated number of dead rows
last_vacuum	timestamp with time zone	Last time when the table is cleared
last_autovacuum	timestamp with time zone	Last time at which the table was vacuumed by the autovacuum daemon
last_analyze	timestamp with time zone	Last time when the table is analyzed
last_autoanalyze	timestamp with time zone	Last time at which the table was analyzed by the autovacuum daemon
vacuum_count	bigint	Number of times the table is cleared
autovacuum_count	bigint	Number of times the table has been vacuumed by the autovacuum daemon

Name	Type	Description
analyze_count	bigint	Number of times the table is analyzed
autoanalyze_count	bigint	Number of times the table has been analyzed by the autovacuum daemon
last_data_changed	timestamp with time zone	Last time at which the table was updated (by **INSERT**/**UPDATE**/ **DELETE** or **EXCHANGE**/ **TRUNCATE**/**DROP** *partition*). This column is recorded only on the local primary database node.

13.3.164 PG_STAT_BAD_BLOCK

PG_STAT_BAD_BLOCK shows statistics about Page or CU verification failures after a node is started.

Table 13-270 PG_STAT_BAD_BLOCK columns

Name	Type	Description
nodename	text	Node name
databaseid	integer	OID of a database
tablespaceid	integer	Tablespace OID
relfilenode	integer	File object ID
bucketid	smallint	ID of the bucket for consistent hashing
forknum	integer	File type The values are as follows: **0**: main data file **1**: FSM file **2**: VM file **3**: BCM file If the value is greater than 4, it indicates a data file of each column in a column-store table.
error_count	integer	Number of verification failures
first_time	timestamp with time zone	Time of the first verification failure

Name	Type	Description
last_time	timestamp with time zone	Time of the latest verification failure

13.3.165 PG_STAT_BGWRITER

PG_STAT_BGWRITER showing statistics about the background writer process's activity.

Table 13-271 PG_STAT_BGWRITER columns

Name	Type	Description
checkpoints_timed	bigint	Number of scheduled checkpoints that have been performed
checkpoints_req	bigint	Number of requested checkpoints that have been performed
checkpoint_write_time	double precision	Total time that has been spent in the portion of checkpoint processing where files are written to disk (unit: ms)
checkpoint_sync_time	double precision	Total time that has been spent in the portion of checkpoint processing where files are synchronized to disk (unit: ms)
buffers_checkpoint	bigint	Number of buffers written during checkpoints
buffers_clean	bigint	Number of buffers written by the background writer
maxwritten_clean	bigint	Number of times the background writer stopped a cleaning scan because it had written too many buffers
buffers_backend	bigint	Number of buffers written directly by a backend
buffers_backend_fsync	bigint	Number of times a backend had to execute its own fsync call (normally the background writer handles those even when the backend does its own write)

Name	Type	Description
buffers_alloc	bigint	Number of buffers allocated
stats_reset	timestamp with time zone	Time at which these statistics were last reset

13.3.166 PG_STAT_DATABASE

PG_STAT_DATABASE contains database statistics for each database in GaussDB.

Table 13-272 PG_STAT_DATABASE columns

Name	Type	Description
datid	oid	OID of a database
datname	name	Name of the database
numbackends	integer	Number of backends currently connected to this database. This is the only column in this view that returns a value reflecting the current state; all other columns return the accumulated values since the last reset.
xact_commit	bigint	Number of transactions in this database that have been committed
xact_rollback	bigint	Number of transactions in this database that have been rolled back
blks_read	bigint	Number of disk blocks read in this database
blks_hit	bigint	Number of times disk blocks were found in the buffer cache (unnecessary as the number includes only hits in the database buffer cache)
tup_returned	bigint	Number of rows returned by queries in this database
tup_fetched	bigint	Number of rows fetched by queries in this database
tup_inserted	bigint	Number of rows inserted by queries in this database
tup_updated	bigint	Number of rows updated by queries in this database
tup_deleted	bigint	Number of rows deleted by queries in this database

Name	Type	Description
conflicts	bigint	Number of queries canceled due to database recovery conflicts (conflicts occurring only on the standby server). For details, see **PG_STAT_DATABASE_CONFLICTS**.
temp_files	bigint	Number of temporary files created by queries in this database. All temporary files are counted, regardless of why the temporary file was created (for example, sorting or hashing), and regardless of the **log_temp_files** setting.
temp_bytes	bigint	Total amount of data written to temporary files by queries in this database. All temporary files are counted, regardless of why the temporary file was created, and regardless of the **log_temp_files** setting.
deadlocks	bigint	Number of deadlocks detected in this database
blk_read_time	double precision	Time spent reading data file blocks by backends in this database (unit: ms)
blk_write_time	double precision	Time spent writing into data file blocks by backends in this database (unit: ms)
stats_reset	timestamp with time zone	Time at which the current statistics were reset

13.3.167 PG_STAT_DATABASE_CONFLICTS

PG_STAT_DATABASE_CONFLICTS displays statistics about database conflicts.

Table 13-273 PG_STAT_DATABASE_CONFLICTS columns

Name	Type	Description
datid	oid	Database ID
datname	name	Database name
confl_tablespace	bigint	Number of conflicting tablespaces
confl_lock	bigint	Number of conflicting locks
confl_snapshot	bigint	Number conflicting snapshots
confl_bufferpin	bigint	Number of conflicting buffers

Name	Type	Description
confl_deadlock	bigint	Number of conflicting deadlocks

13.3.168 PG_STAT_USER_FUNCTIONS

PG_STAT_USER_FUNCTIONS shows user-defined function status information in the namespace. (The language of the function is non-internal language.)

Table 13-274 PG_STAT_USER_FUNCTIONS columns

Name	Type	Description
funcid	oid	OID of a function
schemaname	name	Schema name
funcname	name	Function name
calls	bigint	Number of times the function has been called
total_time	double precision	Total time spent in the function and all other functions called by it
self_time	double precision	Total time spent in the function itself, excluding other functions called by it

13.3.169 PG_STAT_USER_INDEXES

PG_STAT_USER_INDEXES displays information about the index status of user-defined ordinary tables and TOAST tables.

Table 13-275 PG_STAT_USER_INDEXES columns

Name	Type	Description
relid	oid	OID of the table for this index
indexrelid	oid	OID of the index
schemaname	name	Name of the schema that the index is in
relname	name	Name of the table for the index
indexrelname	name	Index name
idx_scan	bigint	Number of index scans initiated on the index
idx_tup_read	bigint	Number of index entries returned by scans on the index

Name	Type	Description
idx_tup_fetch	bigint	Number of live table rows fetched by simple index scans using the index

13.3.170 PG_STAT_USER_TABLES

PG_STAT_USER_TABLES displays information about user-defined ordinary tables and TOAST tables in the namespaces.

Table 13-276 PG_STAT_USER_TABLES columns

Name	Type	Description
relid	oid	OID of the table
schemaname	name	Name of the schema that the table is in
relname	name	Table name
seq_scan	bigint	Number of sequential scans initiated on the table
seq_tup_read	bigint	Number of live rows fetched by sequential scans
idx_scan	bigint	Number of index scans initiated on the table
idx_tup_fetch	bigint	Number of live rows fetched by index scans
n_tup_ins	bigint	Number of rows inserted
n_tup_upd	bigint	Number of rows updated
n_tup_del	bigint	Number of rows deleted
n_tup_hot_upd	bigint	Number of rows HOT updated (with no separate index update required)
n_live_tup	bigint	Estimated number of live rows
n_dead_tup	bigint	Estimated number of dead rows
last_vacuum	timestamp with time zone	Last time at which the table was manually vacuumed (excluding **VACUUM FULL**)
last_autovacuum	timestamp with time zone	Last time at which the table was vacuumed by the autovacuum daemon

Name	Type	Description
last_analyze	timestamp with time zone	Last time at which the table was manually analyzed
last_autoanalyze	timestamp with time zone	Last time at which the table was analyzed by the autovacuum daemon
vacuum_count	bigint	Number of times the table has been manually vacuumed (not counting **VACUUM FULL**)
autovacuum_count	bigint	Number of times the table has been vacuumed by the autovacuum daemon
analyze_count	bigint	Number of times the table has been manually analyzed
autoanalyze_count	bigint	Number of times the table has been analyzed by the autovacuum daemon
last_data_changed	timestamp with time zone	Last modification time of the table data

13.3.171 PG_STAT_REPLICATION

PG_STAT_REPLICATION displays information about log synchronization status, such as the locations of the sender sending logs and the receiver receiving logs.

Table 13-277 PG_STAT_REPLICATION columns

Name	Type	Description
pid	bigint	Process ID of the thread
usesysid	oid	User system ID
usename	name	Username
application_name	text	Program name
client_addr	inet	Client address
client_hostname	text	Client name
client_port	integer	Port of the client
backend_start	timestamp with time zone	Start time of the program

Name	Type	Description
state	text	Log replication state: ● Catch-up state ● Consistent streaming state
sender_sent_location	text	Location where the sender sends logs
receiver_write_location	text	Location where the receiver writes logs
receiver_flush_location	text	Location where the receive end flushes logs
receiver_replay_location	text	Location where the receive end replays logs
sync_priority	integer	Priority of synchronous duplication (**0** indicates asynchronization.)

Name	Type	Description
sync_state	text	Synchronization state ● Asynchronous replication ● Synchronous replication ● Potential synchronization ● Qurom: switches between the synchronous and asynchronous states to ensure that there are more than a certain number of synchronous standby servers. Generally, the number of synchronous standby servers is (n+1)/2-1, where *n* indicates the total number of copies. Whether the standby server is synchronous depends on whether logs are received first. For details, see the description of the **synchronous_standby_names** parameter.

13.3.172 PG_STAT_SUBSCRIPTION

PG_STAT_SUBSCRIPTION displays the detailed synchronization information about the subscription.

Table 13-278 PG_STAT_SUBSCRIPTION columns

Name	Type	Description
subid	oid	Subscription OID.
subname	name	Subscription name.
pid	integer	ID of the background Apply thread.

Name	Type	Description
received_lsn	text	Latest LSA received from the publisher.
last_msg_send_time	timestamp with time zone	Time when the last message is sent.
last_msg_receipt_time	timestamp with time zone	Time when the last message is received by the subscriber.
latest_end_lsn	text	LSN of the sender when the last keepalive message is received.
latest_end_time	timestamp with time zone	Time when the last keepalive message is received.

13.3.173 PG_STAT_SYS_INDEXES

PG_STAT_SYS_INDEXES displays index status information about all the system catalogs in the **pg_catalog** and **information_schema** schemas.

Table 13-279 PG_STAT_SYS_INDEXES columns

Name	Type	Description
relid	oid	OID of the table for this index
indexrelid	oid	OID of the index
schemaname	name	Name of the schema that the index is in
relname	name	Name of the table for the index
indexrelname	name	Index name
idx_scan	bigint	Number of index scans initiated on the index
idx_tup_read	bigint	Number of index entries returned by scans on the index
idx_tup_fetch	bigint	Number of live table rows fetched by simple index scans using the index

13.3.174 PG_STAT_SYS_TABLES

PG_STAT_SYS_TABLES displays statistics about the system catalogs of all the namespaces in **pg_catalog** and **information_schema** schemas.

Table 13-280 PG_STAT_SYS_TABLES columns

Name	Type	Description
relid	oid	OID of a table
schemaname	name	Schema name of the table
relname	name	Table name
seq_scan	bigint	Number of sequential scans initiated on the table
seq_tup_read	bigint	Number of live rows fetched by sequential scans
idx_scan	bigint	Number of index scans initiated on the table
idx_tup_fetch	bigint	Number of live rows fetched by index scans
n_tup_ins	bigint	Number of rows inserted
n_tup_upd	bigint	Number of rows updated
n_tup_del	bigint	Number of rows deleted
n_tup_hot_upd	bigint	Number of rows HOT updated (with no separate index update required)
n_live_tup	bigint	Estimated number of live rows
n_dead_tup	bigint	Estimated number of dead rows
last_vacuum	timestamp with time zone	Last time when the table was manually vacuumed (excluding **VACUUM FULL**)
last_autovacuum	timestamp with time zone	Last time when the table was vacuumed by the autovacuum daemon
last_analyze	timestamp with time zone	Last time when the table was manually analyzed
last_autoanalyze	timestamp with time zone	Last time when the table was analyzed by the autovacuum daemon
vacuum_count	bigint	Number of times the table has been manually vacuumed (not counting **VACUUM FULL**)
autovacuum_count	bigint	Number of times the table has been vacuumed by the autovacuum daemon

Name	Type	Description
analyze_count	bigint	Number of times the table has been manually analyzed
autoanalyze_count	bigint	Number of times the table has been analyzed by the autovacuum daemon
last_data_changed	timestamp with time zone	Last modification time of the table data

13.3.175 PG_STAT_XACT_ALL_TABLES

PG_STAT_XACT_ALL_TABLES displays transaction status information about all ordinary tables and TOAST tables in the namespaces.

Table 13-281 PG_STAT_XACT_ALL_TABLES columns

Name	Type	Description
relid	oid	OID of the table
schemaname	name	Name of the schema that the table is in
relname	name	Table name
seq_scan	bigint	Number of sequential scans initiated on the table
seq_tup_read	bigint	Number of live rows fetched by sequential scans
idx_scan	bigint	Number of index scans initiated on the table
idx_tup_fetch	bigint	Number of live rows fetched by index scans
n_tup_ins	bigint	Number of rows inserted
n_tup_upd	bigint	Number of rows updated
n_tup_del	bigint	Number of rows deleted
n_tup_hot_upd	bigint	Number of rows HOT updated (with no separate index update required)

13.3.176 PG_STAT_XACT_SYS_TABLES

PG_STAT_XACT_SYS_TABLES displays transaction status information of the system catalog in the namespace.

Table 13-282 PG_STAT_XACT_SYS_TABLES columns

Name	Type	Description
relid	oid	OID of the table
schemaname	name	Name of the schema that the table is in
relname	name	Table name
seq_scan	bigint	Number of sequential scans initiated on the table
seq_tup_read	bigint	Number of live rows fetched by sequential scans
idx_scan	bigint	Number of index scans initiated on the table
idx_tup_fetch	bigint	Number of live rows fetched by index scans
n_tup_ins	bigint	Number of rows inserted
n_tup_upd	bigint	Number of rows updated
n_tup_del	bigint	Number of rows deleted
n_tup_hot_upd	bigint	Number of rows HOT updated (with no separate index update required)

13.3.177 PG_STAT_XACT_USER_FUNCTIONS

PG_STAT_XACT_USER_FUNCTIONS contains statistics on the execution of each function.

Table 13-283 PG_STAT_XACT_USER_FUNCTIONS columns

Name	Type	Description
funcid	oid	OID of a function
schemaname	name	Schema name
funcname	name	Function name
calls	bigint	Number of times the function has been called
total_time	double precision	Total time spent in the function and all other functions called by it
self_time	double precision	Total time spent in the function itself, excluding other functions called by it

13.3.178 PG_STAT_XACT_USER_TABLES

PG_STAT_XACT_USER_TABLES displays transaction status information of the user table in the namespace.

Table 13-284 PG_STAT_XACT_USER_TABLES columns

Name	Type	Description
relid	oid	OID of the table
schemaname	name	Name of the schema that the table is in
relname	name	Table name
seq_scan	bigint	Number of sequential scans initiated on the table
seq_tup_read	bigint	Number of live rows fetched by sequential scans
idx_scan	bigint	Number of index scans initiated on the table
idx_tup_fetch	bigint	Number of live rows fetched by index scans
n_tup_ins	bigint	Number of rows inserted
n_tup_upd	bigint	Number of rows updated
n_tup_del	bigint	Number of rows deleted
n_tup_hot_upd	bigint	Number of rows HOT updated (with no separate index update required)

13.3.179 PG_STATIO_ALL_INDEXES

PG_STATIO_ALL_INDEXES contains one row for each index in the current database, showing I/O statistics about accesses to that specific index.

Table 13-285 PG_STATIO_ALL_INDEXES columns

Name	Type	Description
relid	oid	OID of the table for this index
indexrelid	oid	OID of the index
schemaname	name	Name of the schema that the index is in
relname	name	Name of the table that the index is created for
indexrelname	name	Index name
idx_blks_read	bigint	Number of disk blocks read from the index

Name	Type	Description
idx_blks_hit	bigint	Number of cache hits in the index

13.3.180 PG_STATIO_ALL_SEQUENCES

PG_STATIO_ALL_SEQUENCES contains the I/O statistics of each sequence in the current database.

Table 13-286 PG_STATIO_ALL_SEQUENCES columns

Name	Type	Description
relid	oid	OID of this sequence
schemaname	name	Name of the schema where the sequence is in
relname	name	Name of the sequence
blks_read	bigint	Number of disk blocks read from the sequence
blks_hit	bigint	Cache hits in the sequence

13.3.181 PG_STATIO_ALL_TABLES

PG_STATIO_ALL_TABLES contains I/O statistics for each table (including the TOAST table) in the current database.

Table 13-287 PG_STATIO_ALL_TABLES columns

Name	Type	Description
relid	oid	Table OID
schemaname	name	Name of the schema that the table is in
relname	name	Table name
heap_blks_read	bigint	Number of disk blocks read from the table
heap_blks_hit	bigint	Number of cache hits in the table
idx_blks_read	bigint	Number of disk blocks read from all indexes in the table
idx_blks_hit	bigint	Number of cache hits in indexes in the table
toast_blks_read	bigint	Number of disk blocks read from the TOAST table (if any) in the table

Name	Type	Description
toast_blks_hit	bigint	Number of buffer hits in the TOAST table (if any) in the table
tidx_blks_read	bigint	Number of disk blocks read from the TOAST table index (if any) in the table
tidx_blks_hit	bigint	Number of buffer-hits in the TOAST table index (if any) in the table

13.3.182 PG_STATIO_SYS_INDEXES

PG_STATIO_SYS_INDEXES displays I/O status information for all system catalog indexes in a namespace.

Table 13-288 PG_STATIO_SYS_INDEXES columns

Name	Type	Description
relid	oid	OID of the table for this index
indexrelid	oid	OID of the index
schemaname	name	Name of the schema that the index is in
relname	name	Name of the table that the index is created for
indexrelname	name	Index name
idx_blks_read	bigint	Number of disk blocks read from the index
idx_blks_hit	bigint	Number of cache hits in the index

13.3.183 PG_STATIO_SYS_SEQUENCES

PG_STATIO_SYS_SEQUENCES displays I/O status information about all the sequences in the namespace.

Table 13-289 PG_STATIO_SYS_SEQUENCES columns

Name	Type	Description
relid	oid	OID of this sequence
schemaname	name	Name of the schema where the sequence is in
relname	name	Name of the sequence

Name	Type	Description
blks_read	bigint	Number of disk blocks read from the sequence
blks_hit	bigint	Cache hits in the sequence

13.3.184 PG_STATIO_SYS_TABLES

PG_STATIO_SYS_TABLES shows I/O status information about all the system catalogs in the namespace.

Table 13-290 PG_STATIO_SYS_TABLES columns

Name	Type	Description
relid	oid	Table OID
schemaname	name	Name of the schema the table is in
relname	name	Table name
heap_blks_read	bigint	Number of disk blocks read from the table
heap_blks_hit	bigint	Number of cache hits in the table
idx_blks_read	bigint	Number of disk blocks read from all indexes in the table
idx_blks_hit	bigint	Number of cache hits in indexes in the table
toast_blks_read	bigint	Number of disk blocks read from the TOAST table (if any) in the table
toast_blks_hit	bigint	Number of buffer hits in the TOAST table (if any) in the table
tidx_blks_read	bigint	Number of disk blocks read from the TOAST table index (if any) in the table
tidx_blks_hit	bigint	Number of buffer-hits in the TOAST table index (if any) in the table

13.3.185 PG_STATIO_USER_INDEXES

PG_STATIO_USER_INDEXES displays I/O status information about all the user relationship table indexes in the namespace.

Table 13-291 PG_STATIO_USER_INDEXES columns

Name	Type	Description
relid	oid	OID of the table that the index is created for
indexrelid	oid	OID of the index
schemaname	name	Name of the schema that the index is in
relname	name	Name of the table for the index
indexrelname	name	Index name
idx_blks_read	bigint	Number of disk blocks read from the index
idx_blks_hit	bigint	Number of cache hits in the index

13.3.186 PG_STATIO_USER_SEQUENCES

PG_STATIO_USER_SEQUENCES shows I/O status information about all the user relationship table sequences in the namespace.

Table 13-292 PG_STATIO_USER_SEQUENCES columns

Name	Type	Description
relid	oid	OID of this sequence
schemaname	name	Name of the schema where the sequence is in
relname	name	Name of the sequence
blks_read	bigint	Number of disk blocks read from the sequence
blks_hit	bigint	Cache hits in the sequence

13.3.187 PG_STATIO_USER_TABLES

PG_STATIO_USER_TABLES displays I/O status information about all the user relationship tables in the namespace.

Table 13-293 PG_STATIO_USER_TABLES columns

Name	Type	Description
relid	oid	Table OID

Name	Type	Description
schemaname	name	Name of the schema that the table is in
relname	name	Table name
heap_blks_read	bigint	Number of disk blocks read from the table
heap_blks_hit	bigint	Number of cache hits in the table
idx_blks_read	bigint	Number of disk blocks read from all indexes in the table
idx_blks_hit	bigint	Number of cache hits in indexes in the table
toast_blks_read	bigint	Number of disk blocks read from the TOAST table (if any) in the table
toast_blks_hit	bigint	Number of buffer hits in the TOAST table (if any) in the table
tidx_blks_read	bigint	Number of disk blocks read from the TOAST table index (if any) in the table
tidx_blks_hit	bigint	Number of buffer-hits in the TOAST table index (if any) in the table

13.3.188 PG_TABLES

PG_TABLES provides access to each table in the database.

Table 13-294 PG_TABLES columns

Name	Type	Reference	Description
schemaname	name	**PG_NAMESPACE**.nspname	Name of the schema that contains a table
tablename	name	**PG_CLASS**.relname	Table name
tableowner	name	pg_get_userbyid(**PG_CLASS**.relowner)	Table owner
tablespace	name	**PG_TABLESPACE**.spcname	Tablespace that contains the table. The default value is null.
hasindexes	boolean	**PG_CLASS**.relhasindex	The value is **TRUE** if the table has (or recently had) an index; otherwise, the value is **FALSE**.
hasrules	boolean	**PG_CLASS**.relhasruls	The value is **TRUE** if the table has rules; otherwise, the value is **FALSE**.

Name	Type	Reference	Description
hastriggers	boolean	PG_CLASS.RELHASTRIGGERS	The value is **TRUE** if the table has triggers; otherwise, the value is **FALSE**.
tablecreator	name	pg_get_userbyid(po.creator)	Table creator.
created	timestamp with time zone	pg_object.ctime	Creation time of the object
last_ddl_time	timestamp with time zone	pg_object.mtime	Last modification time of the object

13.3.189 PG_TDE_INFO

PG_TDE_INFO provides encryption information of the entire database.

Table 13-295 PG_TDE_INFO columns

Name	Type	Description
is_encrypt	boolean	Encryption status of the database ● **f**: The database is not encrypted. ● **t**: The database is encrypted.
g_tde_algo	text	Encryption algorithm ● SM4-CTR-128 ● AES-CTR-128
remain	text	Reserved column

13.3.190 PG_THREAD_WAIT_STATUS

PG_THREAD_WAIT_STATUS allows you to test the block waiting status about the backend thread and auxiliary thread of the current instance.

Table 13-296 PG_THREAD_WAIT_STATUS columns

Name	Type	Description
node_name	text	Current node name
db_name	text	Database name
thread_name	text	Thread name
query_id	bigint	Query ID. The value of this column is the same as that of **debug_query_id**.
tid	bigint	Current thread ID
sessionid	bigint	Current session ID
lwtid	integer	Lightweight thread ID
psessionid	bigint	Parent session ID
tlevel	integer	Level of the streaming thread
smpid	integer	Concurrent thread ID
wait_status	text	Waiting status of the current thread. For details about the waiting status, see **Table 13-297**.
wait_event	text	If **wait_status** is set to **acquire lock**, **acquire lwlock**, or **wait io**, this column describes the lock, lightweight lock, and I/O information, respectively. If **wait_status** is not any of the three values, this column is empty.
locktag	text	Information about the lock that the current thread is waiting for
lockmode	text	Lock mode that the current thread is waiting to obtain. The values include modes in the table-level lock, row-level lock, and page-level lock.
block_sessionid	bigint	ID of the session that blocks the current thread from obtaining the lock
global_sessionid	text	Global session ID

The waiting statuses in the **wait_status** column are as follows:

Table 13-297 Waiting status list

Value	Description
none	Waiting for no event
acquire lock	Waiting for locking until the locking succeeds or times out

Value	Description
acquire lwlock	Waiting for a lightweight lock
wait io	Waiting for I/O completion
wait cmd	Waiting for reading network communication packets to complete
wait pooler get conn	Waiting for the pooler to obtain the connection
wait pooler abort conn	Waiting for the pooler to terminate the connection
wait pooler clean conn	Waiting for the pooler to clear connections
pooler create conn: [nodename], total N	Waiting for the pooler to set up a connection. The connection is being established with the node specified by *nodename*, and there are *N* connections waiting to be set up.
get conn	Obtaining the connection to other nodes
set cmd: [nodename]	Waiting for running the **SET**, **RESET**, **TRANSACTION BLOCK LEVEL PARA SET**, or **SESSION LEVEL PARA SET** statement on the connection. The statement is being executed on the node specified by *nodename*.
cancel query	Canceling the SQL statement that is being executed through the connection
stop query	Stopping the query that is being executed through the connection
wait node: [nodename](plevel), total N, [phase]	Waiting for receiving data from a connected node. The thread is waiting for data from the plevel thread of the node specified by *nodename*. The data of *N* connections is waiting to be returned. If *phase* is included, the possible phases are as follows: ● **begin**: The transaction is being started. ● **commit**: The transaction is being committed. ● **rollback**: The transaction is being rolled back.
wait transaction sync: xid	Waiting for synchronizing the transaction specified by *xid*

Value	Description
wait wal sync	Waiting for the completion of WAL of synchronization from the specified LSN to the standby instance
wait data sync	Waiting for the completion of data page synchronization to the standby instance
wait data sync queue	Waiting for putting the data pages that are in the row-store or the CU in the column-store into the synchronization queue
flush data: [nodename](plevel), [phase]	Waiting for sending data to the plevel thread of the node specified by *nodename*. If *phase* is included, the possible phase is **wait quota**, indicating that the current communication flow is waiting for the quota value.
stream get conn: [nodename], total N	Waiting for connecting to the consumer object of the node specified by *nodename* when the stream flow is initialized. There are *N* consumers waiting to be connected.
wait producer ready: [nodename](plevel), total N	Waiting for each producer to be ready when the stream flow is initialized. The thread is waiting for the procedure of the plevel thread on the *nodename* node to be ready. There are *N* producers waiting to be ready.
synchronize quit	Waiting for the threads in the stream thread group to quit when the stream plan ends
wait stream nodegroup destroy	Waiting for destroying the stream node group when the stream plan ends
wait active statement	Waiting for job execution under resource and load control
analyze: [relname], [phase]	The thread is doing **ANALYZE** to the *relname* table. If *phase* is included, the possible phase is **autovacuum**, indicating that the database automatically enables the AutoVacuum thread to execute **ANALYZE**.
vacuum: [relname], [phase]	The thread is doing **VACUUM** to the *relname* table. If *phase* is included, the possible phase is **autovacuum**, indicating that the database automatically enables the AutoVacuum thread to execute **VACUUM**.
vacuum full: [relname]	The thread is doing **VACUUM FULL** to the *relname* table.
create index	An index is being created.

Value	Description
HashJoin - [build hash | write file]	The **HashJoin** operator is being executed. In this phase, you need to pay attention to the execution time-consuming. ● **build hash**: The **HashJoin** operator is creating a hash table. ● **write file**: The **HashJoin** operator is writing data to disks.
HashAgg - [build hash | write file]	The **HashAgg** operator is being executed. In this phase, you need to pay attention to the execution time-consuming. ● **build hash**: The **HashAgg** operator is creating a hash table. ● **write file**: The **HashAgg** operator is writing data to disks.
HashSetop - [build hash | write file]	The **HashSetop** operator is being executed. In this phase, you need to pay attention to the execution time-consuming. ● **build hash**: The **HashSetop** operator is creating a hash table. ● **write file**: The **HashSetop** operator is writing data to disks.
Sort | Sort - [fetch tuple | write file]	The **Sort** operator is used for sorting. **fetch tuple** indicates that the **Sort** operator is obtaining tuples, and **write file** indicates that the **Sort** operator is writing data to disks.
Material | Material - write file	The **Material** operator is being executed. **write file** indicates that the **Material** operator is writing data to disks.
NestLoop	The **NestLoop** operator is being executed.
wait memory	Waiting for obtaining the memory
wait sync consumer next step	Waiting for the consumer to execute the stream operator
wait sync producer next step	Waiting for the producer to execute the stream operator
standby read recovery conflict	The read-only mode of the standby node conflicts with the log replay mode.
standby get snapshot	The standby node obtains the snapshot in read-only mode.
wait reserve td	Waiting for the td application

Value	Description
vacuum gpi	Clearing the global partition index in the vacuum or autovacuum process.

If **wait_status** is **acquire lwlock**, **acquire lock**, or **wait io**, there is an event performing I/O operations or waiting for obtaining the corresponding lightweight lock or transaction lock.

The following table describes the corresponding wait events when **wait_status** is **acquire lwlock**. If **wait_event** is **extension**, the lightweight lock is dynamically allocated and is not monitored.

Table 13-298 List of wait events corresponding to lightweight locks

wait_event	Description
ShmemIndexLock	Used to protect the primary index table, a hash table, in shared memory.
OidGenLock	Used to prevent different threads from generating the same OID.
XidGenLock	Used to prevent two transactions from obtaining the same transaction ID.
ProcArrayLock	Used to prevent concurrent access to or concurrent modification on ProcArray shared arrays.
SInvalReadLock	Used to prevent concurrent execution with invalid message deletion.
SInvalWriteLock	Used to prevent concurrent execution with invalid message write and deletion.
WALInsertLock	Used to prevent concurrent execution with WAL insertion.
WALWriteLock	Used to prevent concurrent write from a WAL buffer to a disk.
ControlFileLock	Used to prevent concurrent read/write or concurrent write/write on the **pg_control** file.
CheckpointLock	Used to prevent multi-checkpoint concurrent execution.
CLogControlLock	Used to prevent concurrent access to or concurrent modification on the Clog control data structure.

wait_event	Description
SubtransControlLock	Used to prevent concurrent access to or concurrent modification on the subtransaction control data structure.
MultiXactGenLock	Used to allocate a unique MultiXact ID in serial mode.
MultiXactOffsetControlLock	Used to prevent concurrent read/write or concurrent write/write on **pg_multixact/offset**.
MultiXactMemberControl-Lock	Used to prevent concurrent read/write or concurrent write/write on **pg_multixact/members**.
RelCacheInitLock	Used to add a lock before any operations are performed on the **init** file when messages are invalid.
CheckpointerCommLock	Used to send file flush requests to a checkpointer. The request structure needs to be inserted to a request queue in serial mode.
TwoPhaseStateLock	Used to prevent concurrent access to or modification on two-phase information sharing arrays.
TablespaceCreateLock	Used to check whether a tablespace already exists.
BtreeVacuumLock	Used to prevent **VACUUM** from clearing pages that are being used by B-tree indexes.
AutovacuumLock	Used to access the autovacuum worker array in serial mode.
AutovacuumScheduleLock	Used to distribute tables requiring **VACUUM** in serial mode.
AutoanalyzeLock	Used to obtain and release resources related to a task that allows for autoanalyze execution.
SyncScanLock	Used to determine the start position of a relfilenode during heap scanning.
NodeTableLock	Used to protect a shared structure that stores database node information.
PoolerLock	Used to prevent two threads from simultaneously obtaining the same connection from a connection pool.
RelationMappingLock	Used to wait for the mapping file between system catalogs and storage locations to be updated.

wait_event	Description
AsyncCtlLock	Used to prevent concurrent access to or concurrent modification on the sharing notification status.
AsyncQueueLock	Used to prevent concurrent access to or concurrent modification on the sharing notification queue.
SerializableXactHashLock	Used to prevent concurrent read/write or concurrent write/write on a sharing structure for serializable transactions.
SerializableFinishedListLock	Used to prevent concurrent read/write or concurrent write/write on a shared linked list for completed serial transactions.
SerializablePredicateLockList-Lock	Used to protect a linked list of serializable transactions that have locks.
OldSerXidLock	Used to protect a structure that records serializable transactions that have conflicts.
FileStatLock	Used to protect a data structure that stores statistics file information.
SyncRepLock	Used to protect Xlog synchronization information during primary/standby replication.
DataSyncRepLock	Used to protect data page synchronization information during primary/standby replication.
CStoreColspaceCacheLock	Used to add a lock when CU space is allocated for a column-store table.
CStoreCUCacheSweepLock	Used to add a lock when CU caches used by a column-store table are cyclically washed out.
MetaCacheSweepLock	Used to add a lock when metadata is cyclically washed out.
ExtensionConnectorLibLock	Adds a lock when a specific dynamic library is loaded or uninstalled in ODBC connection initialization scenarios.
SearchServerLibLock	Used to add a lock on the file read operation when a specific dynamic library is initially loaded in GPU-accelerated scenarios.
LsnXlogChkFileLock	Used to serially update the Xlog flush points for primary and standby servers recorded in a specific structure.
ReplicationSlotAllocation-Lock	Used to add a lock when a primary server allocates stream replication slots during primary/standby replication.

wait_event	Description
ReplicationSlotControlLock	Used to prevent concurrent update of stream replication slot status during primary/standby replication.
ResourcePoolHashLock	Used to prevent concurrent access to or concurrent modification on a resource pool table, a hash table.
WorkloadStatHashLock	Used to prevent concurrent access to or concurrent modification on a hash table that contains SQL requests from the primary node of the database.
WorkloadIoStatHashLock	Used to prevent concurrent access to or concurrent modification on a hash table that contains I/O information of the current database node.
WorkloadCGroupHashLock	Used to prevent concurrent access to or concurrent modification on a hash table that contains Cgroup information.
OBSGetPathLock	Used to prevent concurrent read/write or concurrent write/write on an OBS path.
WorkloadUserInfoLock	Used to prevent concurrent access to or concurrent modification on a hash table that contains user information about load management. (The current feature is a lab feature. Contact Huawei technical support before using it.)
WorkloadRecordLock	Used to prevent concurrent access to or concurrent modification on a hash table that contains requests received by the primary node of the database during adaptive memory management.
WorkloadIOUtilLock	Used to protect a structure that records **iostat** and CPU load information.
WorkloadNodeGroupLock	Used to prevent concurrent access to or concurrent modification on a hash table that contains node group information in memory.
JobShmemLock	Used to protect global variables in the shared memory that is periodically read during a scheduled task.
OBSRuntimeLock	Used to obtain environment variables, for example, *GASSHOME*.

wait_event	Description
LLVMDumpIRLock	Used to export the assembly language for dynamically generating functions. The current feature is a lab feature. Contact Huawei technical support before using it.
LLVMParseIRLock	Used to compile and parse a finished IR function from the IR file at the start position of a query. The current feature is a lab feature. Contact Huawei technical support before using it.
CriticalCacheBuildLock	Used to load caches from a shared or local cache initialization file.
WaitCountHashLock	Used to protect a shared structure in user statement counting scenarios.
BufMappingLock	Used to protect operations on a shared-buffer mapping table.
LockMgrLock	Used to protect the information about a common lock structure.
PredicateLockMgrLock	Used to protect the information about a lock structure that has serializable transactions.
OperatorRealTLock	Used to prevent concurrent access to or concurrent modification on a global structure that contains real-time data at the operator level.
OperatorHistLock	Used to prevent concurrent access to or concurrent modification on a global structure that contains historical data at the operator level.
SessionRealTLock	Used to prevent concurrent access to or concurrent modification on a global structure that contains real-time data at the query level.
SessionHistLock	Used to prevent concurrent access to or concurrent modification on a global structure that contains historical data at the query level.
CacheSlotMappingLock	Used to protect global CU cache information.
BarrierLock	Used to ensure that only one thread is creating a barrier at a time.
dummyServerInfoCacheLock	Used to protect a global hash table where the information about database connections is cached.
RPNumberLock	Used by a database node on a computing GaussDB to count the number of threads for a task where plans are being executed.

wait_event	Description
CBMParseXlogLock	Used to protect the lock used when CBM parses Xlogs.
RelfilenodeReuseLock	Used to prevent the link to a reused column attribute file from being canceled by mistake.
RcvWriteLock	Used to prevent concurrent call of **WalDataRcvWrite**.
PercentileLock	Used to protect global percentile buffers.
CSNBufMappingLock	Used to protect CSN pages.
UniqueSQLMappingLock	Used to protect a unique SQL hash table.
DelayDDLLock	Used to prevent concurrent DDL operations.
CLOG Ctl	Used to prevent concurrent access to or concurrent modification on the Clog control data structure.
Async Ctl	Used to protect asynchronization buffers.
MultiXactOffset Ctl	Used to protect SLRU buffers of a MultiXact offset.
MultiXactMember Ctl	Used to protect SLRU buffer of a MultiXact member.
OldSerXid SLRU Ctl	Used to protect SLRU buffers of old transaction IDs.
ReplicationSlotLock	Used to protect a replication slot.
PGPROCLock	Used to protect the PGPROC structure.
MetaCacheLock	Used to protect meta caches.
DataCacheLock	Used to protect data caches.
InstrUserLock	Used to protect a user hash table.
BadBlockStatHashLock	Used to protect the hash table **global_bad_block_stat**
BufFreelistLock	Used to ensure the atomicity of free list operations in the shared buffer.
CUSlotListLock	Used to control concurrent operations on the slot of the column-store buffer.
AddinShmemInitLock	Used to protect the initialization of the shared memory object.
AlterPortLock	Used to protect the coordinator node from changing the registration port number.

wait_event	Description
FdwPartitionCaheLock	Management lock of the buffer of the HDFS partitioned table.
DfsConnectorCacheLock	Management lock of the DFSConnector buffer.
DfsSpaceCacheLock	Management lock of the HDFS tablespace management buffer.
FullBuildXlogCopyStartPtr-Lock	Used to protect Xlog copy operations in the full build.
DfsUserLoginLock	Used for HDFS user login and authentication.
LogicalReplicationSlotPersis-tentDataLock	Used to protect data in the replication slot during logical replication.
WorkloadSessionInfoLock	Used to protect the access to the session info memory hash table of the load management module. (The current feature is a lab feature. Contact Huawei technical support before using it.)
InstrWorkloadLock	Used to protect the access to the memory hash table that stores load management statistics. (The current feature is a lab feature. Contact Huawei technical support before using it.)
PgfdwLock	Used by the management instance to establish a connection to the foreign server.
InstanceTimeLock	Used to obtain time information of sessions in an instance.
XlogRemoveSegLock	Used to protect Xlog segment file recycling.
DnUsedSpaceHashLock	Used to update space usage information corresponding to a session.
CsnMinLock	Used to calculate CSNmin.
GPCCommitLock	Used to protect the addition of the global Plan Cache hash table. The current feature is a lab feature. Contact Huawei technical support before using it.
GPCClearLock	Used to protect the clearing of the global plan cache hash table. The current feature is a lab feature. Contact Huawei technical support before using it.
GPCTimelineLock	Used to protect the timeline check of the global plan cache hash table. The current feature is a lab feature. Contact Huawei technical support before using it.

wait_event	Description
TsTagsCacheLock	Used to manage the time series tag cache.
InstanceRealTLock	Used to protect the update of the hash table that stores shared instance statistics.
CLogBufMappingLock	Used to manage the cache of commit logs.
GPCMappingLock	Used to manage the global plan cache. The current feature is a lab feature. Contact Huawei technical support before using it.
GPCPrepareMappingLock	Used to manage the global plan cache. The current feature is a lab feature. Contact Huawei technical support before using it.
BufferIOLock	Used to protect I/O operations on pages in the shared buffer.
BufferContentLock	Used to protect the read and modification of the page content in the shared buffer.
CSNLOG Ctl	Used for CSN log management.
DoubleWriteLock	Used to manage doublewrite operations.
RowPageReplicationLock	Used to manage data page replication of row-store.
MatviewSeqnoLock	Used to manage the cache of materialized views.
GPRCMappingLock	Used to manage the access and modification operations of the global cache hash table of autonomous transactions.
extension	Other lightweight locks
StartBlockMappingLock	Used by globalstat to obtain information such as startblockarray from pgstat.
PldebugLock	Used to debug stored procedures and perform concurrent maintenance operations.
wait active statement	Waiting for job execution under resource and load control
wait memory	Waiting for obtaining the memory
IOStatLock	Used to concurrently maintain the hash table of resource management I/O statistics.
gtt_shared_ctl	Used to protect global temporary tables to share the concurrent read and write of hash tables.

The following table describes the corresponding wait events when **wait_status** is **wait io**.

Table 13-299 List of wait events corresponding to I/Os

wait_event	Description
BufFileRead	Reads data from a temporary file to a specified buffer.
BufFileWrite	Writes the content of a specified buffer to a temporary file.
ControlFileRead	Reads the **pg_control** file, mainly during database startup, checkpoint execution, and primary/standby verification.
ControlFileSync	Flushes the **pg_control** file to a disk, during database initialization.
ControlFileSyncUpdate	Flushes the **pg_control** file to a disk, mainly during database startup, checkpoint execution, and primary/standby verification.
ControlFileWrite	Writes the **pg_control** file, during database initialization.
ControlFileWriteUpdate	Updates the **pg_control** file, mainly during database startup, checkpoint execution, and primary/standby verification.
CopyFileRead	Reads a file during file copying.
CopyFileWrite	Writes a file during file copying.
DataFileExtend	Writes a file during file extension.
DataFileFlush	Flushes a table data file to a disk.
DataFileImmediateSync	Flushes a table data file to a disk immediately.
DataFilePrefetch	Reads a table data file asynchronously.
DataFileRead	Reads a table data file synchronously.
DataFileSync	Flushes table data file modifications to a disk.
DataFileTruncate	Truncates a table data file.
DataFileWrite	Writes a table data file.
LockFileAddToDataDirRead	Reads the **postmaster.pid** file.
LockFileAddToDataDirSync	Flushes the **postmaster.pid** file to a disk.
LockFileAddToDataDirWrite	Writes PID information into the **postmaster.pid** file.
LockFileCreateRead	Reads the LockFile file **%s.lock**.

wait_event	Description
LockFileCreateSync	Flushes the LockFile file **%s.lock** to a disk.
LockFileCreateWRITE	Writes PID information into the LockFile file **%s.lock**.
RelationMapRead	Reads the mapping file between system catalogs and storage locations.
RelationMapSync	Flushes the mapping file between system catalogs and storage locations to a disk.
RelationMapWrite	Writes the mapping file between system catalogs and storage locations.
ReplicationSlotRead	Reads a stream replication slot file during a restart.
ReplicationSlotRestoreSync	Flushes a stream replication slot file to a disk during a restart.
ReplicationSlotSync	Flushes a temporary stream replication slot file to a disk during checkpoint execution.
ReplicationSlotWrite	Writes a temporary stream replication slot file during checkpoint execution.
SLRUFlushSync	Flushes the **pg_clog**, **pg_subtrans**, and **pg_multixact** files to a disk, mainly during checkpoint execution and database shutdown.
SLRURead	Reads the **pg_clog**, **pg_subtrans**, and **pg_multixact** files.
SLRUSync	Writes dirty pages into the **pg_clog**, **pg_subtrans**, and **pg_multixact** files, and flushes the files to a disk, mainly during checkpoint execution and database shutdown.
SLRUWrite	Writes the **pg_clog**, **pg_subtrans**, and **pg_multixact** files.
TimelineHistoryRead	Reads the timeline history file during database startup.
TimelineHistorySync	Flushes the timeline history file to a disk during database startup.
TimelineHistoryWrite	Writes to the timeline history file during database startup.
TwophaseFileRead	Reads the **pg_twophase** file, mainly during two-phase transaction commit and restoration.
TwophaseFileSync	Flushes the **pg_twophase** file to a disk, mainly during two-phase transaction commit and restoration.

wait_event	Description
TwophaseFileWrite	Writes the **pg_twophase** file, mainly during two-phase transaction commit and restoration.
WALBootstrapSync	Flushes an initialized WAL file to a disk during database initialization.
WALBootstrapWrite	Writes an initialized WAL file during database initialization.
WALCopyRead	Read operation generated when an existing WAL file is read for replication after archiving and restoration.
WALCopySync	Flushes a replicated WAL file to a disk after archiving and restoration.
WALCopyWrite	Write operation generated when an existing WAL file is read for replication after archiving and restoration.
WALInitSync	Flushes a newly initialized WAL file to a disk during log reclaiming or writing.
WALInitWrite	Initializes a newly created WAL file to **0** during log reclaiming or writing.
WALRead	Reads data from Xlogs during redo operations on two-phase files.
WALSyncMethodAssign	Flushes all open WAL files to a disk.
WALWrite	Writes a WAL file.
WALBufferAccess	WAL buffer access (To ensure performance, only the number of access times is counted in the kernel code, and the access duration is not counted.)
WALBufferFull	Writes WAL files when the WAL buffer is full.
DoubleWriteFileRead	Doublewrites and reads a file.
DoubleWriteFileSync	Doublewrites a file and forcibly flushes files to disks.
DoubleWriteFileWrite	Doublewrites a file and writes a file.
PredoProcessPending	Waits for the playback of other records to complete during parallel log playback.
PredoApply	Waits for other threads to play back logs to the LSN of the current thread during parallel log playback.
DisableConnectFileRead	Reads the HA lock fragment logic file.

wait_event	Description
DisableConnectFileSync	Forcibly flushes the HA lock fragment logic file to disks.
DisableConnectFileWrite	Writes the HA lock fragment logic file.

The following table describes the corresponding wait events when **wait_status** is **acquire lock**.

Table 13-300 List of wait events corresponding to transaction locks

wait_event	Description
relation	Adds a lock to a table.
extend	Adds a lock to a table being scaled out.
partition	Adds a lock to a partitioned table.
partition_seq	Adds a lock to a partition of a partitioned table.
page	Adds a lock to a table page.
tuple	Adds a lock to a tuple on a page.
transactionid	Adds a lock to a transaction ID.
virtualxid	Adds a lock to a virtual transaction ID.
object	Adds a lock to an object.
cstore_freespace	Adds a lock to idle column-store space.
userlock	Adds a lock to a user.
advisory	Adds an advisory lock.

13.3.191 PG_TIMEZONE_ABBREVS

PG_TIMEZONE_ABBREVS displays information about all available time zones.

Table 13-301 PG_TIMEZONE_ABBREVS columns

Name	Type	Description
abbrev	text	Time zone name abbreviation
utc_offset	interval	Offset from UTC
is_dst	boolean	Whether DST is used. If DST is used, the value is **TRUE**. Otherwise, the value is **FALSE**.

13.3.192 PG_TIMEZONE_NAMES

PG_TIMEZONE_NAMES displays all time zone names that can be recognized by **SET TIMEZONE**, along with their abbreviations, UTC offsets, and daylight saving time (DST) statuses.

Table 13-302 PG_TIMEZONE_NAMES columns

Name	Type	Description
name	text	Name of the time zone
abbrev	text	Abbreviation of the ime zone name
utc_offset	interval	Offset from UTC
is_dst	boolean	Whether DST is used. If DST is used, the value is **TRUE**. Otherwise, the value is **FALSE**.

13.3.193 PG_TOTAL_MEMORY_DETAIL

PG_TOTAL_MEMORY_DETAIL displays memory usage of a node in the database.

Table 13-303 PG_TOTAL_MEMORY_DETAIL columns

Name	Type	Description
nodename	text	Specifies the node name.
memorytype	text	Memory name
memorymbytes	integer	Size of the used memory in the unit of MB.

13.3.194 PG_TOTAL_USER_RESOURCE_INFO

PG_TOTAL_USER_RESOURCE_INFO displays resource usage of all users. Only administrators can query this view. This view is valid only when **use_workload_manager** is set to **on**. I/O monitoring items are valid only when **enable_logical_io_statistics** is set to **on**.

Table 13-304 PG_TOTAL_USER_RESOURCE_INFO columns

Name	Type	Description
username	name	Username

Name	Type	Description
used_memory	integer	Used memory, in MB
total_memory	integer	Available memory, in MB. The value **0** indicates that the available memory is not limited and depends on the maximum memory available in the database.
used_cpu	double precision	Number of CPU cores in use. CPU usage data is collected only in complex jobs, and the value is the CPU usage of the related Cgroup.
total_cpu	integer	Total number of CPU cores of the Cgroup associated with the user on the node
used_space	bigint	Used permanent table storage space, in KB
total_space	bigint	Available permanent table storage space, in KB (**-1** if the storage space is not limited)
used_temp_space	bigint	Used temporary space, in KB
total_temp_space	bigint	Total available temporary space, in KB (**-1** if the temporary space is not limited)
used_spill_space	bigint	Size of the used operator flushing space, in KB
total_spill_space	bigint	Total size of the available operator flushing space, in KB (**-1** if the space is not limited)
read_kbytes	bigint	Primary database node: total bytes read by the user's complex jobs on all database nodes in the last 5 seconds, in KB Database node: total bytes read by the user's complex jobs from the instance startup time to the current time, in KB
write_kbytes	bigint	Primary database node: total bytes written by the user's complex jobs on all database nodes in the last 5 seconds, in KB Database node: total bytes written by the user's complex jobs from the instance startup time to the current time, in KB
read_counts	bigint	Primary database node: total number of read times of the user's complex jobs on all database nodes in the last 5 seconds Database node: total number of read times of the user's complex jobs from the instance startup time to the current time

Name	Type	Description
write_counts	bigint	Primary database node: total number of write times of the user's complex jobs on all database nodes in the last 5 seconds Database node: total number of write times of the user's complex jobs from the instance startup time to the current time
read_speed	double precision	Primary database node: average read rate of the user's complex jobs on a single database node in the last 5 seconds, in KB/s Database node: average read rate of the user's complex jobs on the database node in the last 5 seconds, in KB/s
write_speed	double precision	Primary database node: average write rate of the user's complex jobs on a single database node in the last 5 seconds, in KB/s Database node: average write rate of the user's complex jobs on the database node in the last 5 seconds, in KB/s

13.3.195 PG_TOTAL_USER_RESOURCE_INFO_OID

PG_TOTAL_USER_RESOURCE_INFO_OID displays resource usage of all users. Only administrators can query this view. This view is valid only when **use_workload_manager** is set to **on**.

Table 13-305 PG_TOTAL_USER_RESOURCE_INFO_OID columns

Name	Type	Description
userid	oid	User ID.
used_memory	integer	Size of the memory being used, in MB.
total_memory	integer	Available memory (unit: MB) The value **0** indicates that the available memory is not limited and depends on the maximum memory available in the database.
used_cpu	double precision	Number of CPU cores in use.
total_cpu	integer	Total number of CPU cores of the Cgroup associated with the user on the node.
used_space	bigint	Used storage space, in KB.
total_space	bigint	Available storage space (unit: KB). The value **-1** indicates that the space is not limited.

Name	Type	Description
used_temp_space	bigint	Used temporary storage space, in KB.
total_temp_space	bigint	Total available temporary space, in KB (**-1** if the temporary space is not limited).
used_spill_space	bigint	Used disk space for spilling, in KB.
total_spill_space	bigint	Total available disk space for spilling, in KB. The value **-1** indicates that the space is not limited.
read_kbytes	bigint	Amount of data read from the disk, in KB.
write_kbytes	bigint	Amount of data written to the disk, in KB.
read_counts	bigint	Number of disk read times.
write_counts	bigint	Number of disk write times.
read_speed	double precision	Disk read rate, in B/ms.
write_speed	double precision	Disk write rate, in B/ms.

13.3.196 PG_USER

PG_USER provides information about database users. By default, only the initial user and users with the **sysadmin** attribute can view the information. Other users can view the information only after being granted permissions.

Table 13-306 PG_USER columns

Name	Type	Description
usename	name	Username
usesysid	oid	ID of this user
usecreatedb	boolean	Whether the user has the permissions to create databases
usesuper	boolean	Whether the user is the initial system administrator with the highest permission
usecatupd	boolean	Whether the user can directly update system catalogs. Only the initial system administrator whose **usesysid** is set to **10** has this permission. It is unavailable for other users.

Name	Type	Description
userepl	boolean	Whether the user has the permissions to duplicate data streams
passwd	text	Encrypted user password. The value is displayed as ********.
valbegin	timestamp with time zone	Start time for account validity (**null** if no start time)
valuntil	timestamp with time zone	End time for account validity (**null** if no end time)
respool	name	Resource pool where the user is in
parent	oid	Parent user OID
spacelimit	text	Storage space of the permanent table
tempspacelimit	text	Storage space of the temporary table
spillspacelimit	text	Operator disk flushing space
useconfig	text[]	Session defaults for runtime configuration variables
nodegroup	name	Name of the logical database associated with the user. If the user does not manage the logical database, this parameter is left blank.
usemonitoradmin	boolean	Whether the user is a monitor administrator ● **t** (true): yes ● **f** (false): no
useoperatoradmin	boolean	Whether the user is an O&M administrator ● **t** (true): yes ● **f** (false): no
usepolicyadmin	boolean	Whether the user is a security policy administrator ● **t** (true): yes ● **f** (false): no

13.3.197 PG_USER_MAPPINGS

PG_USER_MAPPINGS provides access to information about user mappings.

This is essentially a publicly readable view of **PG_USER_MAPPING** that leaves out the options column if the user has no rights to use it. Common users must be authorized to access this view.

Table 13-307 PG_USER_MAPPINGS columns

Name	Type	Reference	Description
umid	oid	**PG_USER_MAPPING**.oid	OID of the user mapping
srvid	oid	**PG_FOREIGN_SERVER**.oid	OID of the foreign server that contains the mapping
srvname	name	**PG_FOREIGN_SERVER**.srvname	Name of the foreign server
umuser	oid	**PG_AUTHID**.oid	OID of the local role being mapped (**0** if the user mapping is public)
usename	name	-	Name of the local user to be mapped
umoptions	text[ ]	-	User mapping specific options. If the current user is the owner of the foreign server, the value is **keyword=value strings**. Otherwise, the value is **null**.

13.3.198 PG_VIEWS

PG_VIEWS provides access to basic information about each view in the database.

Table 13-308 PG_VIEWS columns

Name	Type	Reference	Description
schemaname	name	**PG_NAMESPACE**.nspname	Name of the schema that contains the view
viewname	name	**PG_CLASS**.relname	View name
viewowner	name	**PG_AUTHID**.Erolname	Owner of the view
definition	text	-	Definition of the view

13.3.199 PG_VARIABLE_INFO

PG_VARIABLE_INFO records information about transaction IDs and OIDs of the current node in the database.

Table 13-309 PG_VARIABLE_INFO columns

Name	Type	Description
node_name	text	Node name
next_oid	oid	OID generated next time for the node
next_xid	xid	Transaction ID generated next time for the node
oldest_xid	xid	Oldest transaction ID on the node
xid_vac_limit	xid	Critical point (transaction ID) that triggers forcible autovacuum
oldest_xid_db	oid	OID of the database that has the minimum datafrozenxid on the node
last_extend_csn_logpage	xid	Number of the last extended csnlog page
start_extend_csn_logpage	xid	Number of the page from which csnlog extending starts
next_commit_seqno	xid	CSN generated next time for the node
latest_completed_xid	xid	Latest transaction ID on the node after the transaction commission or rollback
startup_max_xid	xid	Last transaction ID before the node is powered off

13.3.200 PG_WLM_STATISTICS

PG_WLM_STATISTICS displays information about workload management after the task is complete or the exception has been handled. The current feature is a lab feature. Contact Huawei technical support before using it. Only the user with sysadmin permission can query this view.

Table 13-310 PG_WLM_STATISTICS columns

Name	Type	Description
statement	text	Statement executed for exception handling.
block_time	bigint	Block time before the statement is executed.

Name	Type	Description
elapsed_time	bigint	Elapsed time when the statement is executed.
total_cpu_time	bigint	Total time used by the CPU on the database instance when the statement is executed for exception handling.
qualification_time	bigint	Period when the statement checks the skew.
cpu_skew_percent	integer	CPU usage skew on the database instance when the statement is executed for exception handling.
control_group	text	Cgroup used when the statement is executed for exception handling.
status	text	Statement status after statement are executed for exception handling, including: ● **pending**: waiting to be executed. ● **running**: being executed. ● **finished**: finished normally. ● **abort**: terminated unexpectedly.
action	text	Actions when statements are executed for exception handling, including: ● **abort**: terminating the operation. ● **adjust**: executing the Cgroup adjustment operations. Currently, you can only perform the demotion operation. ● **finish**: finished normally.

13.3.201 PGXC_PREPARED_XACTS

PGXC_PREPARED_XACTS displays two-phase transactions in the **prepared** phase. Only users with the **system admin** or **monitor admin** permission can view the information.

Table 13-311 PGXC_PREPARED_XACTS columns

Name	Type	Description
pgxc_prepared_xact	text	Displays the two-phase transaction currently in the **prepared** phase.

13.3.202 PGXC_THREAD_WAIT_STATUS

This view is not supported in centralized scenarios.

13.3.203 PLAN_TABLE

PLAN_TABLE displays plan information collected by **EXPLAIN PLAN**. Plan information is in a session-level lifecycle. After a session exits, the data will be deleted. Data is isolated between sessions and between users.

Table 13-312 PLAN_TABLE columns

Name	Type	Description
statement_id	varchar2(30)	Query tag specified by a user
plan_id	bigint	Query ID
id	int	ID of each operator in a generated plan
operation	varchar2(30)	Operation description of an operator in a plan
options	varchar2(255)	Operation action
object_name	name	Object name corresponding to the operation, which is not the object alias used in the query. The object name is defined by users.
object_type	varchar2(30)	Object type
object_owner	name	Schema to which the object belongs. It is defined by users.
projection	varchar2(4000)	Returned column information
cost	float8	Execution cost estimated by the optimizer for an operator
cardinality	float8	Number of rows estimated by the optimizer for an operator

NOTE

- A valid **object_type** value consists of a relkind type defined in PG_CLASS (**TABLE**, **INDEX**, **SEQUENCE**, **VIEW**, **COMPOSITE TYPE**, or **TOASTVALUE TOAST**) and the rtekind type used in the plan (**SUBQUERY**, **JOIN**, **FUNCTION**, **VALUES**, **CTE**, or **REMOTE_QUERY**).
- For RangeTableEntry (RTE), **object_owner** is the object description used in the plan. Non-user-defined objects do not have **object_owner**.
- Information in the **statement_id**, **object_name**, **object_owner**, and **projection** columns is stored in letter cases specified by users and information in other columns is stored in uppercase.
- **PLAN_TABLE** supports only **SELECT** and **DELETE** and does not support other DML operations.

13.3.204 SYS_DUMMY

SYS_DUMMY is automatically created by the database based on the data dictionary. It has only one text column in only one row for storing expression calculation results. This view is accessible to all users. This view exists in both **PG_CATALOG** and **SYS schema**.

Table 13-313 SYS_DUMMY columns

Name	Type	Description
DUMMY	text	Expression calculation result

14 Schemas

The following table describes the schemas supported by GaussDB.

Table 14-1 Schemas supported by GaussDB

Schema	Description
blockchain	Stores the user history table that is automatically created when a tamper-proof table is created in the ledger database. (The current feature is a lab feature. Contact Huawei technical support before using it.)
cstore	Stores auxiliary tables related to column-store tables, such as CUDesc and Delta tables.
db4ai	Manages data of different versions in AI training.
dbe_perf	Diagnoses performance issues and is also the data source of WDR snapshots. After a database is installed, only the initial user and monitoring administrator have permission to view views and functions in this schema by default.
dbe_pldebugger	Debugs PL/pgSQL functions and stored procedures.
snapshot	Manages data related to WDR snapshots. By default, the initial user or monitoring administrator can access the data.
sqladvisor	Is used for distribution column recommendation and is unavailable in centralized deployment.
sys	Provides the system information view APIs.
pg_catalog	Maintains system catalog information, including system catalogs and all built-in data types, functions, and operators.
pg_toast	Stores large objects (for internal use).
public	Public schema. By default, created tables (and other objects) are automatically put into this schema.

Schema	Description
pkg_service	Manages information about the package service.
pkg_util	Manages information about the package tool.
dbe_raw	Advanced function package dbe_raw, which is used to convert raw data, obtain substrings, and calculate the length.
dbe_session	Advanced function package dbe_session, which is used to set the value of a specified attribute and support user query and verification.
dbe_lob	Advanced function package dbe_lob, which is used to read, write, and copy large files (CLOB/BLOB).
dbe_match	Advanced function package dbe_match, which is used to compare character string similarity.
dbe_task	Advanced function package dbe_task, which is used to schedule job tasks, including submitting tasks, canceling tasks, synchronizing task status, and updating task information, so that the database can periodically execute specific tasks.
dbe_sql	Advanced function package dbe_sql, which is used to execute dynamic SQL statements and construct query and other commands during application running.
dbe_file	Advanced function package dbe_file, which is used to read, copy, write, delete, and rename external database data.
dbe_output	Advanced function package dbe_output, which is used to print output information.
dbe_random	Advanced function package dbe_random, which is used to generate random seeds and random numbers.
dbe_application_info	Advanced function package dbe_application_info, which is used for recording client information.
dbe_utility	Advanced function package dbe_utility, which is used to invoke the debugging tool in a stored procedure, for example, to view error stacks.
dbe_scheduler	Advanced function package dbe_scheduler, which is used to create scheduled tasks and enable the database to periodically execute specified tasks through programs and schedules. You can also perform external database tasks by authorizing and providing certificates.
information_schema	Stores information about objects defined in the current database.
dbe_pldeveloper	Compiles and debugs user stored procedures.

14.1 DBE_PERF Schema

In the **DBE_PERF** schema, views are used to diagnose performance issues and are also the data source of WDR snapshots. After the database is installed, only the initial user has the permission for the **DBE_PERF** schema by default. If the database is upgraded from an earlier version, permissions for the **DBE_PERF** schema are the same as those of the earlier version to ensure forward compatibility. Organization views are divided based on multiple dimensions, such as OS, instance, and memory. These views comply with the following naming rules:

- Views starting with **GLOBAL_**: Request data from database nodes and return the data without processing them.
- Views starting with **SUMMARY_**: Summarize data in the database. In most cases, data from database nodes (sometimes only the primary database node) is processed and collected.
- Views that do not start with **GLOBAL_** or **SUMMARY_**: Local views that do not request data from other database nodes.

14.1.1 OS

14.1.1.1 OS_RUNTIME

OS_RUNTIME displays the running status of the current OS.

Table 14-2 OS_RUNTIME columns

Name	Type	Description
id	integer	ID
name	text	Name of the OS running status
value	numeric	Value of the OS running status
comments	text	Remarks of the OS running status
cumulative	boolean	Whether the value of the OS running status is cumulative

14.1.1.2 GLOBAL_OS_RUNTIME

GLOBAL_OS_RUNTIME provides OS running status information about all normal nodes in the database.

Table 14-3 GLOBAL_OS_RUNTIME columns

Name	Type	Description
node_name	name	Node name
id	integer	ID
name	text	Name of the OS running status
value	numeric	Value of the OS running status
comments	text	Remarks of the OS running status
cumulative	boolean	Whether the value of the OS running status is cumulative

14.1.1.3 OS_THREADS

OS_THREADS provides status information about all threads on the current node.

Table 14-4 OS_THREADS columns

Name	Type	Description
node_name	text	Database process name
pid	bigint	ID of the thread running under the current database process
lwpid	integer	Lightweight thread ID corresponding to **pid**
thread_name	text	Name of the thread corresponding to **pid**
creation_time	timestamp with time zone	Creation time of the thread corresponding to **pid**

14.1.1.4 GLOBAL_OS_THREADS

GLOBAL_OS_THREADS provides status information about threads on all normal nodes in the database.

Table 14-5 GLOBAL_OS_THREADS columns

Name	Type	Description
node_name	text	Node name
pid	bigint	ID of the thread running under the current node process

Name	Type	Description
lwpid	integer	Lightweight thread ID corresponding to **pid**
thread_name	text	Name of the thread corresponding to **pid**
creation_time	timestamp with time zone	Creation time of the thread corresponding to **pid**

14.1.1.5 NODE_NAME

Provide the names of all normal nodes in the database.

Table 14-6 NODE_NAME columns

Name	Type	Description
node_name	name	Node name

14.1.2 Instance

14.1.2.1 INSTANCE_TIME

INSTANCE_TIME records time consumption information on the current database node. The time consumption information is classified into the following types:

- DB_TIME: effective time spent by jobs in multi-core scenarios
- CPU_TIME: CPU time cost.
- EXECUTION_TIME: time spent in the executor.
- PARSE_TIME: time spent on parsing SQL statements
- PLAN_TIME: time spent on generating plans
- REWRITE_TIME: time spent on rewriting SQL statements
- PL_COMPILATION_TIME: compilation time of the PL/pgSQL stored procedure.
- PL_COMPILATION_TIME: compilation time of the PL/pgSQL stored procedure
- NET_SEND_TIME: time spent on the network
- DATA_IO_TIME: time spent on I/Os.

Table 14-7 INSTANCE_TIME columns

Name	Type	Description
stat_id	integer	Statistics ID

Name	Type	Description
stat_name	text	Type name
value	bigint	Time value (unit: μs)

14.1.2.2 GLOBAL_INSTANCE_TIME

GLOBAL_INSTANCE_TIME provides time consumption information about all normal nodes in the database. For details about the time types, see the **INSTANCE_TIME** view.

Table 14-8 GLOBAL_INSTANCE_TIME columns

Name	Type	Description
node_name	name	Node name
stat_id	integer	Statistics ID
stat_name	text	Type name
value	bigint	Duration (unit: μs)

14.1.3 Memory

14.1.3.1 GS_SHARED_MEMORY_DETAIL

SHARED_MEMORY_DETAIL queries the usage information about shared memory contexts on the current node.

Table 14-9 GS_SHARED_MEMORY_DETAIL columns

Name	Type	Description
contextname	text	Name of the memory context
level	smallint	Level of the memory context
parent	text	Name of the parent memory context
totalsize	bigint	Total size of the shared memory (unit: byte)
freesize	bigint	Remaining size of the shared memory (unit: byte)
usedsize	bigint	Used size of the shared memory (unit: byte)

14.1.3.2 GLOBAL_MEMORY_NODE_DETAIL

The memory usage of all normal nodes in the database is displayed.

Table 14-10 GLOBAL_MEMORY_NODE_DETAIL fields

Name	Type	Description
nodename	text	Node name

Name	Type	Description
memorytype	text	Memory name. • **max_process_memory**: memory occupied by the database instance • **process_used_memory**: memory occupied by a process • **max_dynamic_memory**: maximum dynamic memory • **dynamic_used_memory**: used dynamic memory • **dynamic_peak_memory**: dynamic peak value of the memory • **dynamic_used_shrctx**: maximum dynamic shared memory context • **dynamic_peak_shrctx**: dynamic peak value of the shared memory context • **max_shared_memory**: maximum shared memory • **shared_used_memory**: used shared memory • **max_cstore_memory**: maximum memory allowed by the column • **cstore_used_memory**: memory used in column storage • max_sctpcomm_memory: maximum memory that can be used for SCTP TCP proxy communication. • **sctpcomm_used_memory**: used memory for TCP proxy communication • **sctpcomm_peak_memory**: memory peak of TCP proxy communication • **other_used_memory**: other used memory • **gpu_max_dynamic_memory**: maximum dynamic memory of GPU • **gpu_dynamic_used_memory**: used dynamic memory of GPU • **gpu_dynamic_peak_memory**: dynamic peak value of the GPU memory • **pooler_conn_memory**: applied memory in the connection pool • **pooler_freeconn_memory**: memory occupied by idle connections in the connection pool • **storage_compress_memory**: memory used by the storage module for compression • **udf_reserved_memory**: reserved memory for the UDF

Name	Type	Description
memorymbytes	integer	Size of the used memory in the unit of MB.

14.1.3.3 GLOBAL_SHARED_MEMORY_DETAIL

GLOBAL_SHARED_MEMORY_DETAIL is used to query the usage of shared memory contexts on all normal nodes in the database.

Table 14-11 GLOBAL_SHARED_MEMORY_DETAIL columns

Name	Type	Description
node_name	name	Node name
contextname	text	Name of the memory context
level	smallint	Level of the memory context
parent	text	Name of the parent memory context
totalsize	bigint	Total size of the shared memory (unit: byte)
freesize	bigint	Remaining size of the shared memory (unit: byte)
usedsize	bigint	Used size of the shared memory (unit: byte)

14.1.3.4 MEMORY_NODE_DETAIL

MEMORY_NODE_DETAIL displays memory usage of a node in the database.

Table 14-12 MEMORY_NODE_DETAIL columns

Name	Type	Description
nodename	text	Node name

Name	Type	Description
memorytype	text	Memory name ● **max_process_memory**: memory occupied by the GaussDB instance ● **process_used_memory**: memory occupied by a process ● **max_dynamic_memory**: maximum dynamic memory ● **dynamic_used_memory**: used dynamic memory ● **dynamic_peak_memory**: dynamic peak value of the memory ● **dynamic_used_shrctx**: maximum dynamic shared memory context ● **dynamic_peak_shrctx**: dynamic peak value of the shared memory context ● **max_shared_memory**: maximum shared memory ● **shared_used_memory**: used shared memory ● **max_cstore_memory**: maximum memory allowed by the column ● **cstore_used_memory**: memory used in column storage ● **max_sctpcomm_memory**: maximum memory allowed for TCP proxy communication ● **sctpcomm_used_memory**: used memory for TCP proxy communication ● **sctpcomm_peak_memory**: memory peak of TCP proxy communication ● **other_used_memory**: other used memory ● **gpu_max_dynamic_memory**: maximum dynamic GPU memory ● **gpu_dynamic_used_memory**: used dynamic memory of GPU ● **gpu_dynamic_peak_memory**: dynamic peak value of the GPU memory ● **pooler_conn_memory**: applied memory in the connection pool ● **pooler_freeconn_memory**: memory occupied by idle connections in the connection pool ● **storage_compress_memory**: memory used by the storage module for compression ● **udf_reserved_memory**: reserved memory for the UDF

Name	Type	Description
memorymbytes	integer	Size of the used memory in the unit of MB.

14.1.3.5 SHARED_MEMORY_DETAIL

SHARED_MEMORY_DETAIL queries the usage information about shared memory contexts on the current node.

Table 14-13 Table 1 SHARED_MEMORY_DETAIL columns

Name	Type	Description
contextname	text	Name of the memory context
level	smallint	Level of the memory context
parent	text	Name of the parent memory context
totalsize	bigint	Total size of the shared memory (unit: byte)
freesize	bigint	Remaining size of the shared memory (unit: byte)
usedsize	bigint	Used size of the shared memory (unit: byte)

14.1.4 File

14.1.4.1 FILE_IOSTAT

FILE_IOSTAT records statistics about data file I/Os to indicate I/O performance and detect performance problems such as abnormal I/O operations.

Table 14-14 FILE_IOSTAT columns

Name	Type	Description
filenum	oid	File identifier
dbid	oid	Database ID
spcid	oid	Tablespace ID
phyrds	bigint	Number of times of reading physical files

Name	Type	Description
phywrts	bigint	Number of times of writing into physical files
phyblkrd	bigint	Number of times of reading physical file blocks
phyblkwrt	bigint	Number of times of writing into physical file blocks
readtim	bigint	Total duration of reading (unit: μs)
writetim	bigint	Total duration of writing (unit: μs)
avgiotim	bigint	Average duration of reading and writing (unit: μs)
lstiotim	bigint	Duration of the last file reading (unit: μs)
miniotim	bigint	Minimum duration of reading and writing (unit: μs)
maxiowtm	bigint	Maximum duration of reading and writing (unit: μs)

14.1.4.2 SUMMARY_FILE_IOSTAT

SUMMARY_FILE_IOSTAT records statistics about data file I/Os in the database to indicate I/O performance and detect performance problems such as abnormal I/O operations.

Table 14-15 SUMMARY_FILE_IOSTAT columns

Name	Type	Description
filenum	oid	File ID
dbid	oid	Database ID
spcid	oid	Tablespace ID
phyrds	numeric	Number of times of reading physical files
phywrts	numeric	Number of times of writing into physical files
phyblkrd	numeric	Number of times of reading physical file blocks
phyblkwrt	numeric	Number of times of writing into physical file blocks

Name	Type	Description
readtim	numeric	Total duration of reading (unit: μs)
writetim	numeric	Total duration of writing (unit: μs)
avgiotim	bigint	Average duration of reading and writing (unit: μs)
lstiotim	bigint	Duration of the last file reading (unit: μs)
miniotim	bigint	Minimum duration of reading and writing (unit: μs)
maxiowtm	bigint	Maximum duration of reading and writing (unit: μs)

14.1.4.3 GLOBAL_FILE_IOSTAT

GLOBAL_FILE_IOSTAT records statistics about data file I/Os on all nodes.

Table 14-16 GLOBAL_FILE_IOSTAT columns

Name	Type	Description
node_name	name	Node name
filenum	oid	File ID
dbid	oid	Database ID
spcid	oid	Tablespace ID
phyrds	bigint	Number of times of reading physical files
phywrts	bigint	Number of times of writing into physical files
phyblkrd	bigint	Number of times of reading physical file blocks
phyblkwrt	bigint	Number of times of writing into physical file blocks
readtim	bigint	Total duration of reading (unit: μs)
writetim	bigint	Total duration of writing (unit: μs)
avgiotim	bigint	Average duration of reading and writing (unit: μs)
lstiotim	bigint	Duration of the last file reading (unit: μs)

Name	Type	Description
miniotim	bigint	Minimum duration of reading and writing (unit: μs)
maxiowtm	bigint	Maximum duration of reading and writing (unit: μs)

14.1.4.4 FILE_REDO_IOSTAT

FILE_REDO_IOSTAT records statistics about redo logs (WALs) on the current node.

Table 14-17 FILE_REDO_IOSTAT columns

Name	Type	Description
phywrts	bigint	Number of times writing into the WAL buffer
phyblkwrt	bigint	Number of blocks written into the WAL buffer
writetim	bigint	Duration of writing into Xlog files (unit: μs)
avgiotim	bigint	Average duration of writing into Xlog files (unit: μs). **avgiotim** = **writetim**/ **phywrts**
lstiotim	bigint	Duration of the last writing into Xlog files (unit: μs)
miniotim	bigint	Minimum duration of writing into Xlog files (unit: μs)
maxiowtm	bigint	Maximum duration of writing into Xlog files (unit: μs)

14.1.4.5 SUMMARY_FILE_REDO_IOSTAT

SUMMARY_FILE_REDO_IOSTAT records statistics about all redo logs and WALs in the database.

Table 14-18 SUMMARY_FILE_REDO_IOSTAT columns

Name	Type	Description
phywrts	numeric	Number of times writing into the WAL buffer

Name	Type	Description
phyblkwrt	numeric	Number of blocks written into the WAL buffer
writetim	numeric	Duration of writing into Xlog files (unit: µs)
avgiotim	bigint	Average duration of writing into Xlog files (unit: µs). **avgiotim** = **writetim**/ **phywrts**
lstiotim	bigint	Duration of the last writing into Xlog files (unit: µs)
miniotim	bigint	Minimum duration of writing into Xlog files (unit: µs)
maxiowtm	bigint	Maximum duration of writing into Xlog files (unit: µs)

14.1.4.6 GLOBAL_FILE_REDO_IOSTAT

GLOBAL_FILE_REDO_IOSTAT displays statistics about redo logs (WALs) on nodes in the database.

Table 14-19 GLOBAL_FILE_REDO_IOSTAT columns

Name	Type	Description
node_name	name	Node name
phywrts	bigint	Number of times writing into the WAL buffer
phyblkwrt	bigint	Number of blocks written into the WAL buffer
writetim	bigint	Duration of writing into Xlog files (unit: µs)
avgiotim	bigint	Average duration of writing into Xlog files (unit: µs). **avgiotim** = **writetim**/ **phywrts**
lstiotim	bigint	Duration of the last writing into Xlog files (unit: µs)
miniotim	bigint	Minimum duration of writing into Xlog files (unit: µs)
maxiowtm	bigint	Maximum duration of writing into Xlog files (unit: µs)

14.1.4.7 LOCAL_REL_IOSTAT

LOCAL_REL_IOSTAT displays the accumulated I/O status of all data files on the current node.

Table 14-20 LOCAL_REL_IOSTAT columns

Name	Type	Description
phyrds	bigint	Number of times of reading physical files
phywrts	bigint	Number of times of writing into physical files
phyblkrd	bigint	Number of times of reading physical file blocks
phyblkwrt	bigint	Number of times of writing into physical file blocks

14.1.4.8 GLOBAL_REL_IOSTAT

GLOBAL_REL_IOSTAT displays statistics about data file I/Os on all nodes.

Table 14-21 GLOBAL_REL_IOSTAT columns

Name	Type	Description
node_name	name	Node name
phyrds	bigint	Number of times of reading physical files
phywrts	bigint	Number of times of writing into physical files
phyblkrd	bigint	Number of times of reading physical file blocks
phyblkwrt	bigint	Number of times of writing into physical file blocks

14.1.4.9 SUMMARY_REL_IOSTAT

SUMMARY_REL_IOSTAT displays statistics about data file I/Os on all nodes.

Table 14-22 SUMMARY_REL_IOSTAT columns

Name	Type	Description
phyrds	numeric	Number of times of reading physical files
phywrts	numeric	Number of times of writing into physical files
phyblkrd	numeric	Number of times of reading physical file blocks
phyblkwrt	numeric	Number of times of writing into physical file blocks

14.1.5 Object

14.1.5.1 STAT_USER_TABLES

STAT_USER_TABLES displays the status information about user-defined ordinary tables in all namespaces on the current node.

Table 14-23 STAT_USER_TABLES columns

Name	Type	Description
relid	oid	Table OID
schemaname	name	Name of the schema that the table is in
relname	name	Table name
seq_scan	bigint	Number of sequential scans initiated on the table
seq_tup_read	bigint	Number of live rows fetched by sequential scans
idx_scan	bigint	Number of index scans initiated on the table
idx_tup_fetch	bigint	Number of live rows fetched by index scans
n_tup_ins	bigint	Number of rows inserted
n_tup_upd	bigint	Number of rows updated
n_tup_del	bigint	Number of rows deleted
n_tup_hot_upd	bigint	Number of rows HOT updated (with no separate index update required)

Name	Type	Description
n_live_tup	bigint	Estimated number of live rows
n_dead_tup	bigint	Estimated number of dead rows
last_vacuum	timestamp with time zone	Last time at which this table was manually vacuumed (not counting **VACUUM FULL**)
last_autovacuum	timestamp with time zone	Last time at which this table was vacuumed by the autovacuum daemon
last_analyze	timestamp with time zone	Last time at which the table was manually analyzed
last_autoanalyze	timestamp with time zone	Last time at which this table was analyzed by the autovacuum daemon
vacuum_count	bigint	Number of times the table has been manually vacuumed (not counting **VACUUM FULL**)
autovacuum_count	bigint	Number of times this table has been vacuumed by the autovacuum daemon
analyze_count	bigint	Number of times the table has been manually analyzed
autoanalyze_count	bigint	Number of times the table has been analyzed by the autovacuum daemon

14.1.5.2 SUMMARY_STAT_USER_TABLES

SUMMARY_STAT_USER_TABLES displays the status information about user-defined ordinary tables in all namespaces in the database.

Table 14-24 SUMMARY_STAT_USER_TABLES columns

Name	Type	Description
schemaname	name	Name of the schema that contains the table
relname	name	Table name
seq_scan	numeric	Number of sequential scans initiated on this table
seq_tup_read	numeric	Number of live rows fetched by sequential scans

Name	Type	Description
idx_scan	numeric	Number of index scans initiated on the table
idx_tup_fetch	numeric	Number of live rows fetched by index scans
n_tup_ins	numeric	Number of rows inserted
n_tup_upd	numeric	Number of rows updated
n_tup_del	numeric	Number of rows deleted
n_tup_hot_upd	numeric	Number of rows HOT updated (with no separate index update required)
n_live_tup	numeric	Estimated number of live rows
n_dead_tup	numeric	Estimated number of dead rows
last_vacuum	timestamp with time zone	Last time at which this table was manually vacuumed (not counting **VACUUM FULL**)
last_autovacuum	timestamp with time zone	Last time at which this table was vacuumed by the autovacuum daemon
last_analyze	timestamp with time zone	Last time at which this table was manually analyzed
last_autoanalyze	timestamp with time zone	Last time at which this table was analyzed by the autovacuum daemon
vacuum_count	numeric	Number of times the table has been manually vacuumed (not counting **VACUUM FULL**)
autovacuum_count	numeric	Number of times the table has been vacuumed by the autovacuum daemon
analyze_count	numeric	Number of times the table has been manually analyzed
autoanalyze_count	numeric	Number of times the table has been analyzed by the autovacuum daemon

14.1.5.3 GLOBAL_STAT_USER_TABLES

GLOBAL_STAT_USER_TABLES displays the status information about user-defined ordinary tables in all namespaces on each node.

Table 14-25 GLOBAL_STAT_USER_TABLES columns

Name	Type	Description
node_name	name	Node name
relid	oid	OID of the table
schemaname	name	Name of the schema that contains the table
relname	name	Table name
seq_scan	bigint	Number of sequential scans initiated on this table
seq_tup_read	bigint	Number of live rows fetched by sequential scans
idx_scan	bigint	Number of index scans initiated on the table
idx_tup_fetch	bigint	Number of live rows fetched by index scans
n_tup_ins	bigint	Number of rows inserted
n_tup_upd	bigint	Number of rows updated
n_tup_del	bigint	Number of rows deleted
n_tup_hot_upd	bigint	Number of rows HOT updated (with no separate index update required)
n_live_tup	bigint	Estimated number of live rows
n_dead_tup	bigint	Estimated number of dead rows
last_vacuum	timestamp with time zone	Last time at which this table was manually vacuumed (not counting **VACUUM FULL**)
last_autovacuum	timestamp with time zone	Last time at which this table was vacuumed by the autovacuum daemon
last_analyze	timestamp with time zone	Last time at which this table was manually analyzed
last_autoanalyze	timestamp with time zone	Last time at which this table was analyzed by the autovacuum daemon
vacuum_count	bigint	Number of times the table has been manually vacuumed (not counting **VACUUM FULL**)
autovacuum_count	bigint	Number of times the table has been vacuumed by the autovacuum daemon

Name	Type	Description
analyze_count	bigint	Number of times the table has been manually analyzed
autoanalyze_count	bigint	Number of times the table has been analyzed by the autovacuum daemon

14.1.5.4 STAT_USER_INDEXES

STAT_USER_INDEXES displays the status information about the index of user-defined ordinary tables in the current database.

Table 14-26 STAT_USER_INDEXES columns

Name	Type	Description
relid	oid	OID of the table for this index
indexrelid	oid	OID of the index
schemaname	name	Name of the schema that the index is in
relname	name	Name of the table for the index
indexrelname	name	Index name
idx_scan	bigint	Number of index scans initiated on the index
idx_tup_read	bigint	Number of index entries returned by scans on the index
idx_tup_fetch	bigint	Number of live table rows fetched by simple index scans using the index

14.1.5.5 SUMMARY_STAT_USER_INDEXES

SUMMARY_STAT_USER_INDEXES displays the status information about the index of user-defined ordinary tables in all databases in the database.

Table 14-27 SUMMARY_STAT_USER_INDEXES columns

Name	Type	Description
schemaname	name	Name of the schema that the index is in
relname	name	Name of the table for the index
indexrelname	name	Index name

Name	Type	Description
idx_scan	numeric	Number of index scans initiated on the index
idx_tup_read	numeric	Number of index entries returned by scans on the index
idx_tup_fetch	numeric	Number of live table rows fetched by simple index scans using the index

14.1.5.6 GLOBAL_STAT_USER_INDEXES

GLOBAL_STAT_USER_INDEXES displays the status information about the index of user-defined ordinary tables in all databases of each node.

Table 14-28 GLOBAL_STAT_USER_INDEXES columns

Name	Type	Description
node_name	name	Node name
relid	oid	OID of the table for this index
indexrelid	oid	OID of the index
schemaname	name	Name of the schema that the index is in
relname	name	Name of the table for the index
indexrelname	name	Index name
idx_scan	bigint	Number of index scans initiated on the index
idx_tup_read	bigint	Number of index entries returned by scans on the index
idx_tup_fetch	bigint	Number of live table rows fetched by simple index scans using the index

14.1.5.7 STAT_SYS_TABLES

STAT_SYS_TABLES displays statistics about all the system catalogs in the **pg_catalog**, **information_schema**, and **pg_toast** schemas on a single node.

Table 14-29 STAT_SYS_TABLES columns

Name	Type	Description
relid	oid	Table OID

Name	Type	Description
schemaname	name	Name of the schema that the table is in
relname	name	Table name
seq_scan	bigint	Number of sequential scans initiated on the table
seq_tup_read	bigint	Number of live rows fetched by sequential scans
idx_scan	bigint	Number of index scans initiated on the table
idx_tup_fetch	bigint	Number of live rows fetched by index scans
n_tup_ins	bigint	Number of rows inserted
n_tup_upd	bigint	Number of rows updated
n_tup_del	bigint	Number of rows deleted
n_tup_hot_upd	bigint	Number of rows HOT updated (with no separate index update required)
n_live_tup	bigint	Estimated number of live rows
n_dead_tup	bigint	Estimated number of dead rows
last_vacuum	timestamp with time zone	Last time at which this table was manually vacuumed (not counting **VACUUM FULL**)
last_autovacuum	timestamp with time zone	Last time at which this table was vacuumed by the autovacuum daemon
last_analyze	timestamp with time zone	Last time at which this table was manually analyzed
last_autoanalyze	timestamp with time zone	Last time at which this table was analyzed by the autovacuum daemon
vacuum_count	bigint	Number of times this table has been manually vacuumed (not counting VACUUM FULL)
autovacuum_count	bigint	Number of times this table has been vacuumed by the autovacuum daemon
analyze_count	bigint	Number of times this table has been manually analyzed

Name	Type	Description
autoanalyze_count	bigint	Number of times this table has been analyzed by the autovacuum daemon

14.1.5.8 SUMMARY_STAT_SYS_TABLES

SUMMARY_STAT_SYS_TABLES displays statistics about all the system catalogs in the **pg_catalog**, **information_schema**, and **pg_toast** schemas in the database.

Table 14-30 SUMMARY_STAT_SYS_TABLES columns

Name	Type	Description
schemaname	name	Name of the schema that contains the table
relname	name	Table name
seq_scan	numeric	Number of sequential scans initiated on this table
seq_tup_read	numeric	Number of live rows fetched by sequential scans
idx_scan	numeric	Number of index scans initiated on the table
idx_tup_fetch	numeric	Number of live rows fetched by index scans
n_tup_ins	numeric	Number of rows inserted
n_tup_upd	numeric	Number of rows updated
n_tup_del	numeric	Number of rows deleted
n_tup_hot_upd	numeric	Number of rows HOT updated (with no separate index update required)
n_live_tup	numeric	Estimated number of live rows
n_dead_tup	numeric	Estimated number of dead rows
last_vacuum	timestamp with time zone	Last time at which this table was manually vacuumed (not counting **VACUUM FULL**)
last_autovacuum	timestamp with time zone	Last time at which this table was vacuumed by the autovacuum daemon
last_analyze	timestamp with time zone	Last time at which this table was manually analyzed

Name	Type	Description
last_autoanalyze	timestamp with time zone	Last time at which this table was analyzed by the autovacuum daemon
vacuum_count	numeric	Number of times the table has been manually vacuumed (not counting **VACUUM FULL**)
autovacuum_count	numeric	Number of times the table has been vacuumed by the autovacuum daemon
analyze_count	numeric	Number of times the table has been manually analyzed
autoanalyze_count	numeric	Number of times the table has been analyzed by the autovacuum daemon

14.1.5.9 GLOBAL_STAT_SYS_TABLES

GLOBAL_STAT_SYS_TABLES displays statistics about all the system catalogs in the **pg_catalog**, **information_schema**, and **pg_toast** schemas on each node.

Table 14-31 GLOBAL_STAT_SYS_TABLES columns

Name	Type	Description
node_name	name	Node name
relid	oid	Table OID
schemaname	name	Name of the schema that contains the table
relname	name	Table name
seq_scan	bigint	Number of sequential scans initiated on this table
seq_tup_read	bigint	Number of live rows fetched by sequential scans
idx_scan	bigint	Number of index scans initiated on the table
idx_tup_fetch	bigint	Number of live rows fetched by index scans
n_tup_ins	bigint	Number of rows inserted
n_tup_upd	bigint	Number of rows updated

Name	Type	Description
n_tup_del	bigint	Number of rows deleted
n_tup_hot_upd	bigint	Number of rows HOT updated (with no separate index update required)
n_live_tup	bigint	Estimated number of live rows
n_dead_tup	bigint	Estimated number of dead rows
last_vacuum	timestamp with time zone	Last time at which this table was manually vacuumed (not counting **VACUUM FULL**)
last_autovacuum	timestamp with time zone	Last time at which this table was vacuumed by the autovacuum daemon
last_analyze	timestamp with time zone	Last time at which this table was manually analyzed
last_autoanalyze	timestamp with time zone	Last time at which this table was analyzed by the autovacuum daemon
vacuum_count	bigint	Number of times the table has been manually vacuumed (not counting **VACUUM FULL**)
autovacuum_count	bigint	Number of times the table has been vacuumed by the autovacuum daemon
analyze_count	bigint	Number of times the table has been manually analyzed
autoanalyze_count	bigint	Number of times the table has been analyzed by the autovacuum daemon

14.1.5.10 STAT_SYS_INDEXES

STAT_SYS_INDEXES displays index status information about all the system catalogs in the **pg_catalog**, **information_schema**, and **pg_toast** schemas.

Table 14-32 STAT_SYS_INDEXES columns

Name	Type	Description
relid	oid	OID of the table for this index
indexrelid	oid	OID of the index
schemaname	name	Name of the schema that the index is in

Name	Type	Description
relname	name	Name of the table that the index is created for
indexrelname	name	Index name
idx_scan	bigint	Number of index scans initiated on the index
idx_tup_read	bigint	Number of index entries returned by scans on the index
idx_tup_fetch	bigint	Number of live table rows fetched by simple index scans using the index

14.1.5.11 SUMMARY_STAT_SYS_INDEXES

SUMMARY_STAT_SYS_INDEXES displays index status information about all the system catalogs in the **pg_catalog**, **information_schema**, and **pg_toast** schemas in the database.

Table 14-33 SUMMARY_STAT_SYS_INDEXES columns

Name	Type	Description
schemaname	name	Name of the schema that the index is in
relname	name	Name of the table for the index
indexrelname	name	Index name
idx_scan	numeric	Number of index scans initiated on the index
idx_tup_read	numeric	Number of index entries returned by scans on the index
idx_tup_fetch	numeric	Number of live table rows fetched by simple index scans using the index

14.1.5.12 GLOBAL_STAT_SYS_INDEXES

GLOBAL_STAT_SYS_INDEXES displays index status information about all the system catalogs in the **pg_catalog**, **information_schema**, and **pg_toast** schemas on each node.

Table 14-34 GLOBAL_STAT_SYS_INDEXES columns

Name	Type	Description
node_name	name	Node name

Name	Type	Description
relid	oid	OID of the table for this index
indexrelid	oid	OID of the index
schemaname	name	Name of the schema that the index is in
relname	name	Name of the table for the index
indexrelname	name	Index name
idx_scan	bigint	Number of index scans initiated on the index
idx_tup_read	bigint	Number of index entries returned by scans on the index
idx_tup_fetch	bigint	Number of live table rows fetched by simple index scans using the index

14.1.5.13 STAT_ALL_TABLES

STAT_ALL_TABLES displays statistics about one row for each table (including TOAST tables) in databases on this node.

Table 14-35 STAT_ALL_TABLES columns

Name	Type	Description
relid	oid	Table OID
schemaname	name	Name of the schema that the table is in
relname	name	Table name
seq_scan	bigint	Number of sequential scans initiated on the table
seq_tup_read	bigint	Number of live rows fetched by sequential scans
idx_scan	bigint	Number of index scans initiated on the table
idx_tup_fetch	bigint	Number of live rows fetched by index scans
n_tup_ins	bigint	Number of rows inserted
n_tup_upd	bigint	Number of rows updated
n_tup_del	bigint	Number of rows deleted

Name	Type	Description
n_tup_hot_upd	bigint	Number of rows HOT updated (with no separate index update required)
n_live_tup	bigint	Estimated number of live rows
n_dead_tup	bigint	Estimated number of dead rows
last_vacuum	timestamp with time zone	Last time at which this table was manually vacuumed (not counting **VACUUM FULL**)
last_autovacuum	timestamp with time zone	Last time at which this table was vacuumed by the autovacuum daemon
last_analyze	timestamp with time zone	Last time at which the table was manually analyzed
last_autoanalyze	timestamp with time zone	Last time at which this table was analyzed by the autovacuum daemon
vacuum_count	bigint	Number of times the table has been manually vacuumed (not counting **VACUUM FULL**)
autovacuum_count	bigint	Number of times this table has been vacuumed by the autovacuum daemon
analyze_count	bigint	Number of times the table has been manually analyzed
autoanalyze_count	bigint	Number of times the table has been analyzed by the autovacuum daemon

14.1.5.14 SUMMARY_STAT_ALL_TABLES

SUMMARY_STAT_ALL_TABLES displays statistics about one row for each table (including TOAST tables) in a database in the entire database system.

Table 14-36 SUMMARY_STAT_ALL_TABLES columns

Name	Type	Description
schemaname	name	Name of the schema that contains the table
relname	name	Table name
seq_scan	numeric	Number of sequential scans initiated on this table
seq_tup_read	numeric	Number of live rows fetched by sequential scans

Name	Type	Description
idx_scan	numeric	Number of index scans initiated on the table
idx_tup_fetch	numeric	Number of live rows fetched by index scans
n_tup_ins	numeric	Number of rows inserted
n_tup_upd	numeric	Number of rows updated
n_tup_del	numeric	Number of rows deleted
n_tup_hot_upd	numeric	Number of rows HOT updated (with no separate index update required)
n_live_tup	numeric	Estimated number of live rows
n_dead_tup	numeric	Estimated number of dead rows
last_vacuum	timestamp with time zone	Last time at which this table was manually vacuumed (not counting **VACUUM FULL**)
last_autovacuum	timestamp with time zone	Last time at which this table was vacuumed by the autovacuum daemon
last_analyze	timestamp with time zone	Last time at which this table was manually analyzed
last_autoanalyze	timestamp with time zone	Last time at which this table was analyzed by the autovacuum daemon
vacuum_count	numeric	Number of times the table has been manually vacuumed (not counting **VACUUM FULL**)
autovacuum_count	numeric	Number of times the table has been vacuumed by the autovacuum daemon
analyze_count	numeric	Number of times the table has been manually analyzed
autoanalyze_count	numeric	Number of times the table has been analyzed by the autovacuum daemon

14.1.5.15 GLOBAL_STAT_ALL_TABLES

GLOBAL_STAT_ALL_TABLES displays statistics about one row for each table (including TOAST tables) in databases on each node.

Table 14-37 GLOBAL_STAT_ALL_TABLES columns

Name	Type	Description
node_name	name	Node name
relid	oid	OID of the table
schemaname	name	Name of the schema that contains the table
relname	name	Table name
seq_scan	bigint	Number of sequential scans initiated on this table
seq_tup_read	bigint	Number of live rows fetched by sequential scans
idx_scan	bigint	Number of index scans initiated on the table
idx_tup_fetch	bigint	Number of live rows fetched by index scans
n_tup_ins	bigint	Number of rows inserted
n_tup_upd	bigint	Number of rows updated
n_tup_del	bigint	Number of rows deleted
n_tup_hot_upd	bigint	Number of rows HOT updated (with no separate index update required)
n_live_tup	bigint	Estimated number of live rows
n_dead_tup	bigint	Estimated number of dead rows
last_vacuum	timestamp with time zone	Last time at which this table was manually vacuumed (not counting **VACUUM FULL**)
last_autovacuum	timestamp with time zone	Last time at which this table was vacuumed by the autovacuum daemon
last_analyze	timestamp with time zone	Last time at which this table was manually analyzed
last_autoanalyze	timestamp with time zone	Last time at which this table was analyzed by the autovacuum daemon
vacuum_count	bigint	Number of times the table has been manually vacuumed (not counting **VACUUM FULL**)
autovacuum_count	bigint	Number of times the table has been vacuumed by the autovacuum daemon

Name	Type	Description
analyze_count	bigint	Number of times the table has been manually analyzed
autoanalyze_count	bigint	Number of times the table has been analyzed by the autovacuum daemon

14.1.5.16 STAT_ALL_INDEXES

STAT_ALL_INDEXES contains every row of each index in databases on the current node, showing statistics about accesses to specific indexes.

Table 14-38 STAT_ALL_INDEXES columns

Name	Type	Description
relid	oid	OID of the table for this index
indexrelid	oid	OID of the index
schemaname	name	Name of the schema that the index is in
relname	name	Name of the table for the index
indexrelname	name	Index name
idx_scan	bigint	Number of index scans initiated on the index
idx_tup_read	bigint	Number of index entries returned by scans on the index
idx_tup_fetch	bigint	Number of live table rows fetched by simple index scans using the index

14.1.5.17 SUMMARY_STAT_ALL_INDEXES

SUMMARY_STAT_ALL_INDEXES displays access statistics about each index in the GaussDB database.

Table 14-39 SUMMARY_STAT_ALL_INDEXES columns

Name	Type	Description
schemaname	name	Name of the schema that the index is in
relname	name	Name of the table for the index
indexrelname	name	Index name

Name	Type	Description
idx_scan	numeric	Number of index scans initiated on the index
idx_tup_read	numeric	Number of index entries returned by scans on the index
idx_tup_fetch	numeric	Number of live table rows fetched by simple index scans using the index

14.1.5.18 GLOBAL_STAT_ALL_INDEXES

GLOBAL_STAT_ALL_INDEXES contains every row of each index in databases on each node, showing statistics about accesses to specific indexes.

Table 14-40 GLOBAL_STAT_ALL_INDEXES columns

Name	Type	Description
node_name	name	Node name
relid	oid	OID of the table for this index
indexrelid	oid	OID of the index
schemaname	name	Name of the schema that the index is in
relname	name	Name of the table for the index
indexrelname	name	Index name
idx_scan	bigint	Number of index scans initiated on the index
idx_tup_read	bigint	Number of index entries returned by scans on the index
idx_tup_fetch	bigint	Number of live table rows fetched by simple index scans using the index

14.1.5.19 STAT_DATABASE

STAT_DATABASE contains statistics for each database on this node.

Table 14-41 STAT_DATABASE columns

Name	Type	Description
datid	oid	OID of a database
datname	name	Database name

Name	Type	Description
numbackends	integer	Number of backends currently connected to this database. This is the only column in this view that returns a value reflecting the current state; all other columns return the accumulated values since the last reset.
xact_commit	bigint	Number of transactions in this database that have been committed
xact_rollback	bigint	Number of transactions in this database that have been rolled back
blks_read	bigint	Number of disk blocks read in this database
blks_hit	bigint	Number of times disk blocks were found in the buffer cache (unnecessary as the number includes only hits in the database buffer cache)
tup_returned	bigint	Number of rows returned by queries in this database
tup_fetched	bigint	Number of rows fetched by queries in this database
tup_inserted	bigint	Number of rows inserted by queries in this database
tup_updated	bigint	Number of rows updated by queries in this database
tup_deleted	bigint	Number of rows deleted by queries in this database
conflicts	bigint	Number of queries canceled due to database recovery conflicts (conflicts occurring only on the standby server). For details, see **STAT_DATABASE_CONFLICTS**.
temp_files	bigint	Number of temporary files created by queries in this database. All temporary files are counted, regardless of why the temporary file was created (for example, sorting or hashing), and regardless of the **log_temp_files** setting.
temp_bytes	bigint	Total amount of data written to temporary files by queries in this database. All temporary files are counted, regardless of why the temporary file was created, and regardless of the **log_temp_files** setting.
deadlocks	bigint	Number of deadlocks detected in this database

Name	Type	Description
blk_read_time	double precision	Time spent reading data file blocks by backends in this database (unit: ms)
blk_write_time	double precision	Time spent reading data file blocks by backends in this database (unit: ms)
stats_reset	timestamp with time zone	Time at which the current statistics were reset

14.1.5.20 SUMMARY_STAT_DATABASE

SUMMARY_STAT_DATABASE contains every row of each database in the database, showing database-wide statistics.

Table 14-42 SUMMARY_STAT_DATABASE

Name	Type	Description
datname	name	Name of the database
numbackends	bigint	Number of backends currently connected to this database. This is the only column in this view that returns a value reflecting the current state; all other columns return the accumulated values since the last reset.
xact_commit	numeric	Number of transactions in this database that have been committed
xact_rollback	numeric	Number of transactions in this database that have been rolled back
blks_read	numeric	Number of disk blocks read in this database
blks_hit	numeric	Number of times disk blocks were found already in the cache, so that a read was not necessary (this only includes hits in the GaussDB cache, not the operating system's file system cache)
tup_returned	numeric	Number of rows returned by queries in this database
tup_fetched	numeric	Number of rows fetched by queries in this database
tup_inserted	bigint	Number of rows inserted by queries in this database

Name	Type	Description
tup_updated	bigint	Number of rows updated by queries in this database
tup_deleted	bigint	Number of rows deleted by queries in this database
conflicts	bigint	Number of queries canceled due to database recovery conflicts (conflicts occurring only on the standby server). For details, see **STAT_DATABASE_CONFLICTS**.
temp_files	numeric	Number of temporary files created by queries in this database. All temporary files are counted, regardless of why the temporary file was created (for example, sorting or hashing), and regardless of the **log_temp_files** setting.
temp_bytes	numeric	Total amount of data written to temporary files by queries in this database. All temporary files are counted, regardless of why the temporary file was created, and regardless of the **log_temp_files** setting.
deadlocks	bigint	Number of deadlocks detected in this database
blk_read_time	double precision	Time spent reading data file blocks by backends in this database (unit: ms)
blk_write_time	double precision	Time spent writing into data file blocks by backends in this database (unit: ms)
stats_reset	timestamp with time zone	Time at which the current statistics were reset

14.1.5.21 GLOBAL_STAT_DATABASE

GLOBAL_STAT_DATABASE contains every row of each database on each node in the entire database system, showing database-wide statistics.

Table 14-43 GLOBAL_STAT_DATABASE columns

Name	Type	Description
node_name	name	Node name
datid	oid	OID of the database
datname	name	Name of the database

Name	Type	Description
numbackends	integer	Number of backends currently connected to this database. This is the only column in this view that returns a value reflecting the current state; all other columns return the accumulated values since the last reset.
xact_commit	bigint	Number of transactions in this database that have been committed
xact_rollback	bigint	Number of transactions in this database that have been rolled back
blks_read	bigint	Number of disk blocks read in this database
blks_hit	bigint	Number of times disk blocks were found in the buffer cache (unnecessary as the number includes only hits in the database kernel buffer cache)
tup_returned	bigint	Number of rows returned by queries in this database
tup_fetched	bigint	Number of rows fetched by queries in this database
tup_inserted	bigint	Number of rows inserted by queries in this database
tup_updated	bigint	Number of rows updated by queries in this database
tup_deleted	bigint	Number of rows deleted by queries in this database
conflicts	bigint	Number of queries canceled due to database recovery conflicts (conflicts occurring only on the standby server). For details, see **STAT_DATABASE_CONFLICTS**.
temp_files	bigint	Number of temporary files created by queries in this database. All temporary files are counted, regardless of why the temporary file was created (for example, sorting or hashing), and regardless of the **log_temp_files** setting.
temp_bytes	bigint	Total amount of data written to temporary files by queries in this database. All temporary files are counted, regardless of why the temporary file was created, and regardless of the **log_temp_files** setting.

Name	Type	Description
deadlocks	bigint	Number of deadlocks detected in this database
blk_read_time	double precision	Time spent reading data file blocks by backends in this database (unit: ms)
blk_write_time	double precision	Time spent writing into data file blocks by backends in this database (unit: ms)
stats_reset	timestamp with time zone	Time at which the current statistics were reset

14.1.5.22 STAT_DATABASE_CONFLICTS

STAT_DATABASE_CONFLICTS displays statistics about database conflicts on the current node.

Table 14-44 STAT_DATABASE_CONFLICTS columns

Name	Type	Description
datid	oid	Database ID
datname	name	Database name
confl_tablespace	bigint	Number of conflicting tablespaces
confl_lock	bigint	Number of conflicting locks
confl_snapshot	bigint	Number of conflicting snapshots
confl_bufferpin	bigint	Number of conflicting buffers
confl_deadlock	bigint	Number of conflicting deadlocks

14.1.5.23 SUMMARY_STAT_DATABASE_CONFLICTS

SUMMARY_STAT_DATABASE_CONFLICTS displays statistics about database conflicts in the database.

Table 14-45 SUMMARY_STAT_DATABASE_CONFLICTS columns

Name	Type	Description
datname	name	Database name

Name	Type	Description
confl_tablespace	bigint	Number of conflicting tablespaces
confl_lock	bigint	Number of conflicting locks
confl_snapshot	bigint	Number of conflicting snapshots
confl_bufferpin	bigint	Number of conflicting buffers
confl_deadlock	bigint	Number of conflicting deadlocks

14.1.5.24 GLOBAL_STAT_DATABASE_CONFLICTS

GLOBAL_STAT_DATABASE_CONFLICTS displays statistics about database conflicts on each node.

Table 14-46 GLOBAL_STAT_DATABASE_CONFLICTS columns

Name	Type	Description
node_name	name	Node name
datid	oid	Database ID
datname	name	Database name
confl_tablespace	bigint	Number of conflicting tablespaces
confl_lock	bigint	Number of conflicting locks
confl_snapshot	bigint	Number of conflicting snapshots
confl_bufferpin	bigint	Number of conflicting buffers
confl_deadlock	bigint	Number of conflicting deadlocks

14.1.5.25 STAT_XACT_ALL_TABLES

STAT_XACT_ALL_TABLES displays transaction status information about all ordinary tables and TOAST tables in the current namespace.

Table 14-47 STAT_XACT_ALL_TABLES columns

Name	Type	Description
relid	oid	Table OID

Name	Type	Description
schemaname	name	Name of the schema that the table is in
relname	name	Table name
seq_scan	bigint	Number of sequential scans initiated on the table
seq_tup_read	bigint	Number of live rows fetched by sequential scans
idx_scan	bigint	Number of index scans initiated on the table
idx_tup_fetch	bigint	Number of live rows fetched by index scans
n_tup_ins	bigint	Number of rows inserted
n_tup_upd	bigint	Number of rows updated
n_tup_del	bigint	Number of rows deleted
n_tup_hot_upd	bigint	Number of rows HOT updated (with no separate index update required)

14.1.5.26 SUMMARY_STAT_XACT_ALL_TABLES

SUMMARY_STAT_XACT_ALL_TABLES displays transaction status information about all ordinary tables and TOAST tables in all namespaces in the database.

Table 14-48 SUMMARY_STAT_XACT_ALL_TABLES columns

Name	Type	Description
schemaname	name	Name of the schema that contains the table
relname	name	Table name
seq_scan	numeric	Number of sequential scans initiated on this table
seq_tup_read	numeric	Number of live rows fetched by sequential scans
idx_scan	numeric	Number of index scans initiated on the table
idx_tup_fetch	numeric	Number of live rows fetched by index scans
n_tup_ins	numeric	Number of rows inserted
n_tup_upd	numeric	Number of rows updated
n_tup_del	numeric	Number of rows deleted
n_tup_hot_upd	numeric	Number of rows HOT updated (with no separate index update required)

14.1.5.27 GLOBAL_STAT_XACT_ALL_TABLES

GLOBAL_STAT_XACT_ALL_TABLES displays transaction status information about all ordinary tables and TOAST tables in namespaces on each node.

Table 14-49 GLOBAL_STAT_XACT_ALL_TABLES columns

Name	Type	Description
node_name	name	Node name
relid	oid	OID of the table
schemaname	name	Name of the schema that contains the table
relname	name	Table name
seq_scan	bigint	Number of sequential scans initiated on this table
seq_tup_read	bigint	Number of live rows fetched by sequential scans
idx_scan	bigint	Number of index scans initiated on the table
idx_tup_fetch	bigint	Number of live rows fetched by index scans
n_tup_ins	bigint	Number of rows inserted
n_tup_upd	bigint	Number of rows updated
n_tup_del	bigint	Number of rows deleted
n_tup_hot_upd	bigint	Number of rows HOT updated (with no separate index update required)

14.1.5.28 STAT_XACT_SYS_TABLES

STAT_XACT_SYS_TABLES displays transaction status information about the system catalogs in namespaces on the current node.

Table 14-50 STAT_XACT_SYS_TABLES columns

Name	Type	Description
relid	oid	Table OID
schemaname	name	Name of the schema that the table is in
relname	name	Table name
seq_scan	bigint	Number of sequential scans initiated on the table

Name	Type	Description
seq_tup_read	bigint	Number of live rows fetched by sequential scans
idx_scan	bigint	Number of index scans initiated on the table
idx_tup_fetch	bigint	Number of live rows fetched by index scans
n_tup_ins	bigint	Number of rows inserted
n_tup_upd	bigint	Number of rows updated
n_tup_del	bigint	Number of rows deleted
n_tup_hot_upd	bigint	Number of rows HOT updated (with no separate index update required)

14.1.5.29 SUMMARY_STAT_XACT_SYS_TABLES

SUMMARY_STAT_XACT_SYS_TABLES displays transaction status information about the system catalogs in namespaces in the database.

Table 14-51 SUMMARY_STAT_XACT_SYS_TABLES columns

Name	Type	Description
schemaname	name	Name of the schema that contains the table
relname	name	Table name
seq_scan	numeric	Number of sequential scans initiated on this table
seq_tup_read	numeric	Number of live rows fetched by sequential scans
idx_scan	numeric	Number of index scans initiated on the table
idx_tup_fetch	numeric	Number of live rows fetched by index scans
n_tup_ins	numeric	Number of rows inserted
n_tup_upd	numeric	Number of rows updated
n_tup_del	numeric	Number of rows deleted
n_tup_hot_upd	numeric	Number of rows HOT updated (with no separate index update required)

14.1.5.30 GLOBAL_STAT_XACT_SYS_TABLES

GLOBAL_STAT_XACT_SYS_TABLES displays transaction status information about the system catalogs in namespaces on each node.

Table 14-52 GLOBAL_STAT_XACT_SYS_TABLES columns

Name	Type	Description
node_name	name	Node name
relid	oid	OID of the table
schemaname	name	Name of the schema that contains the table
relname	name	Table name
seq_scan	bigint	Number of sequential scans initiated on this table
seq_tup_read	bigint	Number of live rows fetched by sequential scans
idx_scan	bigint	Number of index scans initiated on the table
idx_tup_fetch	bigint	Number of live rows fetched by index scans
n_tup_ins	bigint	Number of rows inserted
n_tup_upd	bigint	Number of rows updated
n_tup_del	bigint	Number of rows deleted
n_tup_hot_upd	bigint	Number of rows HOT updated (with no separate index update required)

14.1.5.31 STAT_XACT_USER_TABLES

STAT_XACT_USER_TABLES displays transaction status information about the user tables in namespaces on the current node.

Table 14-53 STAT_XACT_USER_TABLES columns

Name	Type	Description
relid	oid	Table OID
schemaname	name	Name of the schema that the table is in
relname	name	Table name
seq_scan	bigint	Number of sequential scans initiated on the table
seq_tup_read	bigint	Number of live rows fetched by sequential scans
idx_scan	bigint	Number of index scans initiated on the table
idx_tup_fetch	bigint	Number of live rows fetched by index scans

Name	Type	Description
n_tup_ins	bigint	Number of rows inserted
n_tup_upd	bigint	Number of rows updated
n_tup_del	bigint	Number of rows deleted
n_tup_hot_upd	bigint	Number of rows HOT updated (with no separate index update required)

14.1.5.32 SUMMARY_STAT_XACT_USER_TABLES

SUMMARY_STAT_XACT_USER_TABLES displays the transaction status information about the user tables in namespaces in the database.

Table 14-54 SUMMARY_STAT_XACT_USER_TABLES columns

Name	Type	Description
schemaname	name	Name of the schema that contains the table
relname	name	Table name
seq_scan	numeric	Number of sequential scans initiated on this table
seq_tup_read	numeric	Number of live rows fetched by sequential scans
idx_scan	numeric	Number of index scans initiated on the table
idx_tup_fetch	numeric	Number of live rows fetched by index scans
n_tup_ins	numeric	Number of rows inserted
n_tup_upd	numeric	Number of rows updated
n_tup_del	numeric	Number of rows deleted
n_tup_hot_upd	numeric	Number of rows HOT updated (with no separate index update required)

14.1.5.33 GLOBAL_STAT_XACT_USER_TABLES

GLOBAL_STAT_XACT_USER_TABLES displays transaction status information about the user tables in namespaces on each node.

Table 14-55 GLOBAL_STAT_XACT_USER_TABLES columns

Name	Type	Description
node_name	name	Node name
relid	oid	OID of the table

Name	Type	Description
schemaname	name	Name of the schema that contains the table
relname	name	Table name
seq_scan	bigint	Number of sequential scans initiated on this table
seq_tup_read	bigint	Number of live rows fetched by sequential scans
idx_scan	bigint	Number of index scans initiated on the table
idx_tup_fetch	bigint	Number of live rows fetched by index scans
n_tup_ins	bigint	Number of rows inserted
n_tup_upd	bigint	Number of rows updated
n_tup_del	bigint	Number of rows deleted
n_tup_hot_upd	bigint	Number of rows HOT updated (with no separate index update required)

14.1.5.34 STAT_XACT_USER_FUNCTIONS

STAT_XACT_USER_FUNCTIONS displays statistics about function executions in the current transaction.

Table 14-56 STAT_XACT_USER_FUNCTIONS columns

Name	Type	Description
funcid	oid	OID of a function
schemaname	name	Schema name
funcname	name	Function name
calls	bigint	Number of times the function has been called
total_time	double precision	Total time spent in the function and all other functions called by it
self_time	double precision	Total time spent in this function, excluding other functions called by it

14.1.5.35 SUMMARY_STAT_XACT_USER_FUNCTIONS

SUMMARY_STAT_XACT_USER_FUNCTIONS displays statistics about function executions in transactions in the database.

Table 14-57 SUMMARY_STAT_XACT_USER_FUNCTIONS columns

Name	Type	Description
schemaname	name	Schema name
funcname	name	Function name
calls	numeric	Number of times that the function has been called
total_time	double precision	Total time spent in the function and all other functions called by it
self_time	double precision	Total time spent in the function itself, excluding other functions called by it

14.1.5.36 GLOBAL_STAT_XACT_USER_FUNCTIONS

GLOBAL_STAT_XACT_USER_FUNCTIONS displays statistics about function executions in transactions on each node.

Table 14-58 GLOBAL_STAT_XACT_USER_FUNCTIONS columns

Name	Type	Description
node_name	name	Node name
funcid	oid	OID of a function
schemaname	name	Schema name
funcname	name	Function name
calls	bigint	Number of times that the function has been called
total_time	double precision	Total time spent in this function and all other functions called by it
self_time	double precision	Total time spent in this function, excluding other functions called by it

14.1.5.37 STAT_BAD_BLOCK

STAT_BAD_BLOCK displays information about table and index read failures on the current node.

Table 14-59 STAT_BAD_BLOCK columns

Name	Type	Description
nodename	text	Database process name
databaseid	integer	OID of the database
tablespaceid	integer	OID of the tablespace
relfilenode	integer	File node of this relation
bucketid	smallint	ID of the bucket for consistent hashing
forknum	integer	Fork number
error_count	integer	Number of errors
first_time	timestamp with time zone	Time when the first bad block occurred
last_time	timestamp with time zone	Time when the last bad block occurred

14.1.5.38 SUMMARY_STAT_BAD_BLOCK

SUMMARY_STAT_BAD_BLOCK displays information about table and index read failures in the database.

Table 14-60 SUMMARY_STAT_BAD_BLOCK columns

Name	Type	Description
databaseid	integer	OID of the database
tablespaceid	integer	OID of the tablespace
relfilenode	integer	File node of this relation
forknum	bigint	Fork number
error_count	bigint	Number of errors
first_time	timestamp with time zone	Time when the first bad block occurred
last_time	timestamp with time zone	Time when the last bad block occurred

14.1.5.39 GLOBAL_STAT_BAD_BLOCK

GLOBAL_STAT_BAD_BLOCK displays information about table and index read failures on each node.

Table 14-61 GLOBAL_STAT_BAD_BLOCK columns

Name	Type	Description
node_name	text	Node name
databaseid	integer	OID of the database
tablespaceid	integer	OID of the tablespace
relfilenode	integer	File node of this relation
forknum	integer	Fork number
error_count	integer	Number of errors
first_time	timestamp with time zone	Time when the first bad block occurred
last_time	timestamp with time zone	Time when the last bad block occurred

14.1.5.40 STAT_USER_FUNCTIONS

STAT_USER_FUNCTIONS displays user-defined function status information in the current namespace. (The language of the function is non-internal language.)

Table 14-62 STAT_USER_FUNCTIONS columns

Name	Type	Description
funcid	oid	OID of the function
schemaname	name	Schema name
funcname	name	Rename the customized function.
calls	bigint	Number of times the function has been called
total_time	double precision	Total time spent in this function, including other functions called by it (unit: ms)
self_time	double precision	Time spent in this function, excluding other functions called by it (unit: ms)

14.1.5.41 SUMMARY_STAT_USER_FUNCTIONS

SUMMARY_STAT_USER_FUNCTIONS displays statistics about user-defined functions on all database nodes.

Table 14-63 SUMMARY_STAT_USER_FUNCTIONS columns

Name	Type	Description
schemaname	name	Schema name
funcname	name	Function name
calls	numeric	Number of times that the function has been called
total_time	double precision	Total time spent in this function and all other functions called by it (unit: ms)
self_time	double precision	Time spent in this function, excluding other functions called by it (unit: ms)

14.1.5.42 GLOBAL_STAT_USER_FUNCTIONS

GLOBAL_STAT_USER_FUNCTIONS displays statistics about user-defined functions on each node in the database.

Table 14-64 GLOBAL_STAT_USER_FUNCTIONS columns

Name	Type	Description
node_name	name	Node name
funcid	oid	ID of the function
schemaname	name	Schema name
funcname	name	Function name
calls	bigint	Number of times the function has been called
total_time	double precision	Total time spent in this function and all other functions called by it (unit: ms)
self_time	double precision	Total time spent in this function, excluding other functions called by it (unit: ms)

14.1.6 Workload

14.1.6.1 WORKLOAD_SQL_COUNT

WORKLOAD_SQL_COUNT displays the distribution of SQL statements in workloads on the current node. Common users can view only the distribution of SQL statements executed by themselves in workloads, whereas the initial user can view the overall load status of workloads.

Table 14-65 WORKLOAD_SQL_COUNT columns

Name	Type	Description
workload	name	Workload name
select_count	bigint	Number of **SELECT** statements
update_count	bigint	Number of **UPDATE** statements
insert_count	bigint	Number of **INSERT** statements
delete_count	bigint	Number of **DELETE** statements
ddl_count	bigint	Number of **DDL** statements
dml_count	bigint	Number of **DML** statements
dcl_count	bigint	Number of **DCL** statements

14.1.6.2 SUMMARY_WORKLOAD_SQL_COUNT

SUMMARY_WORKLOAD_SQL_COUNT displays the distribution of SQL statements in workloads on the primary database node in the database.

Table 14-66 SUMMARY_WORKLOAD_SQL_COUNT columns

Name	Type	Description
node_name	name	Database process name
workload	name	Workload name
select_count	bigint	Number of **SELECT** statements
update_count	bigint	Number of **UPDATE** statements
insert_count	bigint	Number of **INSERT** statements
delete_count	bigint	Number of **DELETE** statements
ddl_count	bigint	Number of **DDL** statements
dml_count	bigint	Number of **DML** statements
dcl_count	bigint	Number of **DCL** statements

14.1.6.3 WORKLOAD_TRANSACTION

WORKLOAD_TRANSACTION displays information about transactions loaded on the current node.

Table 14-67 WORKLOAD_TRANSACTION columns

Name	Type	Description
workload	name	Workload name
commit_counter	bigint	Number of user transactions committed
rollback_counter	bigint	Number of user transactions rolled back
resp_min	bigint	Minimum response time of user transactions (unit: μs)
resp_max	bigint	Maximum response time of user transactions (unit: μs)
resp_avg	bigint	Average response time of user transactions (unit: μs)
resp_total	bigint	Total response time of user transactions (unit: μs)
bg_commit_counter	bigint	Number of background transactions committed
bg_rollback_counter	bigint	Number of background transactions rolled back
bg_resp_min	bigint	Minimum response time of background transactions (unit: μs)
bg_resp_max	bigint	Maximum response time of background transactions (unit: μs)
bg_resp_avg	bigint	Average response time of background transactions (unit: μs)
bg_resp_total	bigint	Total response time of background transactions (unit: μs)

14.1.6.4 SUMMARY_WORKLOAD_TRANSACTION

SUMMARY_WORKLOAD_TRANSACTION displays information about transactions loaded in the database.

Table 14-68 SUMMARY_WORKLOAD_TRANSACTION columns

Name	Type	Description
workload	name	Workload name

Name	Type	Description
commit_counter	numeric	Number of user transactions committed
rollback_counter	numeric	Number of user transactions rolled back
resp_min	bigint	Minimum response time of user transactions (unit: μs)
resp_max	bigint	Maximum response time of user transactions (unit: μs)
resp_avg	bigint	Average response time of user transactions (unit: μs)
resp_total	numeric	Total response time of user transactions (unit: μs)
bg_commit_counter	numeric	Number of background transactions committed
bg_rollback_counter	numeric	Number of background transactions rolled back
bg_resp_min	bigint	Minimum response time of background transactions (unit: μs)
bg_resp_max	bigint	Maximum response time of background transactions (unit: μs)
bg_resp_avg	bigint	Average response time of background transactions (unit: μs)
bg_resp_total	numeric	Total response time of background transactions (unit: μs)

14.1.6.5 GLOBAL_WORKLOAD_TRANSACTION

GLOBAL_WORKLOAD_TRANSACTION displays load information about workloads on each node.

Table 14-69 GLOBAL_WORKLOAD_TRANSACTION columns

Name	Type	Description
node_name	name	Node name
workload	name	Workload name
commit_counter	bigint	Number of user transactions committed
rollback_counter	bigint	Number of user transactions rolled back
resp_min	bigint	Minimum response time of user transactions (unit: μs)

Name	Type	Description
resp_max	bigint	Maximum response time of user transactions (unit: μs)
resp_avg	bigint	Average response time of user transactions (unit: μs)
resp_total	bigint	Total response time of user transactions (unit: μs)
bg_commit_counter	bigint	Number of background transactions committed
bg_rollback_counter	bigint	Number of background transactions rolled back
bg_resp_min	bigint	Minimum response time of background transactions (unit: μs)
bg_resp_max	bigint	Maximum response time of background transactions (unit: μs)
bg_resp_avg	bigint	Average response time of background transactions (unit: μs)
bg_resp_total	bigint	Total response time of background transactions (unit: μs)

14.1.6.6 WORKLOAD_SQL_ELAPSE_TIME

WORKLOAD_SQL_ELAPSE_TIME collects statistics about SUIDs in workloads.

Table 14-70 WORKLOAD_SQL_ELAPSE_TIME columns

Name	Type	Description
workload	name	Workload name
total_select_elapse	bigint	Total response time of **SELECT** statements (unit: μs)
max_select_elapse	bigint	Maximum response time of **SELECT** statements (unit: μs)
min_select_elapse	bigint	Minimum response time of **SELECT** statements (unit: μs)
avg_select_elapse	bigint	Average response time of **SELECT** statements (unit: μs)
total_update_elapse	bigint	Total response time of **UPDATE** statements (unit: μs)

Name	Type	Description
max_update_elapse	bigint	Maximum response time of **UPDATE** statements (unit: μs)
min_update_elapse	bigint	Minimum response time of **UPDATE** statements (unit: μs)
avg_update_elapse	bigint	Average response time of **UPDATE** statements (unit: μs)
total_insert_elapse	bigint	Total response time of **INSERT** statements (unit: μs)
max_insert_elapse	bigint	Maximum response time of **INSERT** statements (unit: μs)
min_insert_elapse	bigint	Minimum response time of **INSERT** statements (unit: μs)
avg_insert_elapse	bigint	Average response time of **INSERT** statements (unit: μs)
total_delete_elapse	bigint	Total response time of **DELETE** statements (unit: μs)
max_delete_elapse	bigint	Maximum response time of **DELETE** statements (unit: μs)
min_delete_elapse	bigint	Minimum response time of **DELETE** statements (unit: μs)
avg_delete_elapse	bigint	Average response time of **DELETE** statements (unit: μs)

14.1.6.7 SUMMARY_WORKLOAD_SQL_ELAPSE_TIME

SUMMARY_WORKLOAD_SQL_ELAPSE_TIME collects statistics about SUIDs in workloads on the primary database node.

Table 14-71 SUMMARY_WORKLOAD_SQL_ELAPSE_TIM columns

Name	Type	Description
node_name	name	Database process name
workload	name	Workload name
total_select_elapse	bigint	Total response time of **SELECT** statements (unit: μs)
max_select_elapse	bigint	Maximum response time of **SELECT** statements (unit: μs)

Name	Type	Description
min_select_elapse	bigint	Minimum response time of **SELECT** statements (unit: μs)
avg_select_elapse	bigint	Average response time of **SELECT** statements (unit: μs)
total_update_elapse	bigint	Total response time of **UPDATE** statements (unit: μs)
max_update_elapse	bigint	Maximum response time of **UPDATE** statements (unit: μs)
min_update_elapse	bigint	Minimum response time of **UPDATE** statements (unit: μs)
avg_update_elapse	bigint	Average response time of **UPDATE** statements (unit: μs)
total_insert_elapse	bigint	Total response time of **INSERT** statements (unit: μs)
max_insert_elapse	bigint	Maximum response time of **INSERT** statements (unit: μs)
min_insert_elapse	bigint	Minimum response time of **INSERT** statements (unit: μs)
avg_insert_elapse	bigint	Average response time of **INSERT** statements (unit: μs)
total_delete_elapse	bigint	Total response time of **DELETE** statements (unit: μs)
max_delete_elapse	bigint	Maximum response time of **DELETE** statements (unit: μs)
min_delete_elapse	bigint	Minimum response time of **DELETE** statements (unit: μs)
avg_delete_elapse	bigint	Average response time of **DELETE** statements (unit: μs)

14.1.6.8 USER_TRANSACTION

USER_TRANSACTION collects statistics about transactions executed by users. Common users can view only transactions executed by themselves, whereas user **monadmin** can view transactions executed by all users.

Table 14-72 USER_TRANSACTION columns

Name	Type	Description
username	name	Username

Name	Type	Description
commit_counter	bigint	Number of user transactions committed
rollback_counter	bigint	Number of user transactions rolled back
resp_min	bigint	Minimum response time of user transactions (unit: μs)
resp_max	bigint	Maximum response time of user transactions (unit: μs)
resp_avg	bigint	Average response time of user transactions (unit: μs)
resp_total	bigint	Total response time of user transactions (unit: μs)
bg_commit_counter	bigint	Number of background transactions committed
bg_rollback_counter	bigint	Number of background transactions rolled back
bg_resp_min	bigint	Minimum response time of background transactions (unit: μs)
bg_resp_max	bigint	Maximum response time of background transactions (unit: μs)
bg_resp_avg	bigint	Average response time of background transactions (unit: μs)
bg_resp_total	bigint	Total response time of background transactions (unit: μs)

14.1.6.9 GLOBAL_USER_TRANSACTION

GLOBAL_USER_TRANSACTION collects statistics about transactions executed by all users.

Table 14-73 GLOBAL_USER_TRANSACTION columns

Name	Type	Description
node_name	name	Node name
username	name	Username
commit_counter	bigint	Number of user transactions committed
rollback_counter	bigint	Number of user transactions rolled back
resp_min	bigint	Minimum response time of user transactions (unit: μs)

Name	Type	Description
resp_max	bigint	Maximum response time of user transactions (unit: μs)
resp_avg	bigint	Average response time of user transactions (unit: μs)
resp_total	bigint	Total response time of user transactions (unit: μs)
bg_commit_counter	bigint	Number of background transactions committed
bg_rollback_counter	bigint	Number of background transactions rolled back
bg_resp_min	bigint	Minimum response time of background transactions (unit: μs)
bg_resp_max	bigint	Maximum response time of background transactions (unit: μs)
bg_resp_avg	bigint	Average response time of background transactions (unit: μs)
bg_resp_total	bigint	Total response time of background transactions (unit: μs)

14.1.7 Session/Thread

14.1.7.1 SESSION_STAT

SESSION_STAT collects statistics about session status on the current node based on session threads or the **AutoVacuum** thread.

Table 14-74 SESSION_STAT columns

Name	Type	Description
sessid	text	Thread start time and ID
statid	integer	Statistics ID
statname	text	Name of the statistics session
statunit	text	Unit of the statistics session
value	bigint	Value of the statistics session

14.1.7.2 GLOBAL_SESSION_STAT

GLOBAL_SESSION_STAT collects statistics about session status on each node based on session threads or the **AutoVacuum** thread.

Table 14-75 GLOBAL_SESSION_STAT columns

Name	Type	Description
node_name	name	Node name
sessid	text	Thread start time and ID
statid	integer	Statistics ID
statname	text	Name of the statistics session
statunit	text	Unit of the statistics session
value	bigint	Value of the statistics session

14.1.7.3 SESSION_TIME

SESSION_TIME collects statistics about the running time of session threads and time consumed in each execution phase on the current node.

Table 14-76 SESSION_TIME columns

Name	Type	Description
sessid	text	Thread start time and ID
stat_id	integer	Statistics ID
stat_name	text	Session type name
value	bigint	Session value

14.1.7.4 GLOBAL_SESSION_TIME

GLOBAL_SESSION_TIME collects statistics about the running time of session threads and time consumed in each execution phase on each node.

Table 14-77 GLOBAL_SESSION_TIME columns

Name	Type	Description
node_name	name	Node name
sessid	text	Thread start time and ID
stat_id	integer	Statistics ID

Name	Type	Description
stat_name	text	Session type
value	bigint	Session value

14.1.7.5 SESSION_MEMORY

SESSION_MEMORY collects statistics about memory usage at the session level in the unit of MB, including all the memory allocated to GaussDB and stream threads on DNs for jobs currently executed by users.

Table 14-78 SESSION_MEMORY columns

Name	Type	Description
sessid	text	Thread start time and ID
init_mem	integer	Memory allocated to the currently executed job before the job enters the executor
used_mem	integer	Memory allocated to the currently executed job
peak_mem	integer	Peak memory allocated to the currently executed job

14.1.7.6 GLOBAL_SESSION_MEMORY

GLOBAL_SESSION_MEMORY collects statistics about memory usage at the session level on each node in the unit of MB, including all the memory allocated to GaussDB and stream threads on DNs for jobs currently executed by users.

Table 14-79 GLOBAL_SESSION_MEMORY columns

Name	Type	Description
node_name	name	Node name
sessid	text	Thread start time and ID
init_mem	integer	Memory allocated to the currently executed job before the job enters the executor
used_mem	integer	Memory allocated to the currently executed job
peak_mem	integer	Peak memory allocated to the currently executed job

14.1.7.7 SESSION_MEMORY_DETAIL

SESSION_MEMORY_DETAIL collects statistics about thread memory usage by MemoryContext node.

Table 14-80 SESSION_MEMORY_DETAIL columns

Name	Type	Description
sessid	text	Thread start time and ID
sesstype	text	Thread name
contextname	text	Name of the memory context
level	smallint	Level of memory context importance
parent	text	Name of the parent memory context
totalsize	bigint	Size of the allocated memory (unit: byte)
freesize	bigint	Size of the idle memory (unit: byte)
usedsize	bigint	Size of the used memory (unit: byte)

14.1.7.8 GLOBAL_SESSION_MEMORY_DETAIL

GLOBAL_SESSION_MEMORY_DETAIL collects statistics about thread memory usage on each node by MemoryContext node.

Table 14-81 GLOBAL_SESSION_MEMORY_DETAIL columns

Name	Type	Description
node_name	name	Node name
sessid	text	Thread start time and ID
sesstype	text	Thread name
contextname	text	Name of the memory context
level	smallint	Level of memory context importance
parent	text	Name of the parent memory context
totalsize	bigint	Size of the allocated memory (unit: byte)
freesize	bigint	Size of the idle memory (unit: byte)
usedsize	bigint	Size of the used memory (unit: byte)

14.1.7.9 SESSION_STAT_ACTIVITY

SESSION_STAT_ACTIVITY displays information about threads that are running on the current node.

Table 14-82 SESSION_STAT_ACTIVITY columns

Name	Type	Description
datid	oid	OID of the database that the user session connects to in the backend
datname	name	Name of the database that the user session connects to in the backend
pid	bigint	Thread ID of the backend
usesysid	oid	OID of the user logged in to the backend
usename	name	Name of the user logged in to the backend
application_name	text	Name of the application connected to the backend
client_addr	inet	IP address of the client connected to the backend. If this column is null, it indicates either the client is connected via a Unix socket on the server or this is an internal process, such as autovacuum.
client_hostname	text	Host name of the connected client, as reported by a reverse DNS lookup of **client_addr**. This column will be non-null only for IP connections and only when **log_hostname** is enabled.
client_port	integer	TCP port number that the client uses for communication with this backend (**-1** if a Unix socket is used)
backend_start	timestampwith time zone	Time when this process was started, that is, when the client connected to the server
xact_start	timestampwith time zone	Time when current transaction was started (null if no transaction is active). If the current query is the first of its transaction, the value of this column is the same as that of the **query_start** column.
query_start	timestampwith time zone	Time when the currently active query was started, or time when the last query was started if **state** is not **active**
state_change	timestampwith time zone	Time when the **state** was last changed

Name	Type	Description
waiting	boolean	Whether the backend is currently waiting on a lock. If yes, the value is **true**.
enqueue	text	Unsupported currently.

state	text	Overall status of this backend. The value must be one of the following:
		● **active**: The backend is executing a query.
		● **idle**: The backend is waiting for a new client command.
		● **idle in transaction**: The backend is in a transaction, but is not currently executing a query.
		● **idle in transaction (aborted)**: This state is similar to **idle in transaction**, except that one of the statements in the transaction caused an error.
		● **fastpath function call**: The backend is executing a fast-path function.
		● **disabled**: This state is reported if **track_activities** is disabled in this backend.
		NOTE
		Common users can view their own session status only. The state information of other accounts is empty. For example, after the **judy** user is connected to the database, the state information of the **joe** user and the initial user **omm** in **pg_stat_activity** is empty.
		`openGauss=# SELECT datname, usename, usesysid,state,pid FROM pg_stat_activity;`
		`datname    | usename | usesysid | state    |       pid`
		`----------+---------+----------+--------`
		`+-----------------postgres  | omm       | 10`
		`|       |139968752121616`
		`postgres  | omm       | 10       |        |`
		`139968903116560`
		`db_tpcds  | judy      | 16398    | active |`
		`139968391403280`
		`postgres  | omm       | 10       |          |`
		`139968643069712`
		`postgres  | omm       | 10       |        |`
		`139968680818448`
		`postgres  | joe       | 16390    |        |`
		`139968563377936`
		`(6 rows)`

resource_pool	name	Resource pool used by the user
query_id	bigint	ID of a query

Name	Type	Description
query	text	Latest query at the backend. If **state** is **active**, this column shows the ongoing query. In all other states, it shows the last query that was executed.
unique_sql_id	bigint	Unique SQL statement ID.
trace_id	text	Driver-specific trace ID, which is associated with an application request.

14.1.7.10 GLOBAL_SESSION_STAT_ACTIVITY

GLOBAL_SESSION_STAT_ACTIVITY displays information about threads that are running on each node in the database.

Table 14-83 GLOBAL_SESSION_STAT_ACTIVITY columns

Name	Type	Description
coorname	text	Database process name
datid	oid	OID of the database that the user session connects to in the backend
datname	text	Name of the database that the user session connects to in the backend
pid	bigint	Thread ID of the backend
usesysid	oid	OID of the user logged in to the backend
usename	text	Name of the user logged in to the backend
application_name	text	Name of the application connected to the backend
client_addr	inet	IP address of the client connected to the backend. If this column is null, it indicates either the client is connected via a Unix socket on the server or this is an internal process, such as autovacuum.
client_hostname	text	Host name of the connected client, as reported by a reverse DNS lookup of **client_addr**. This column will be non-null only for IP connections and only when **log_hostname** is enabled.
client_port	integer	TCP port number that the client uses for communication with this backend (**-1** if a Unix socket is used)

Name	Type	Description
backend_start	timestampwith time zone	Time when this process was started, that is, when the client connected to the server
xact_start	timestampwith time zone	Time when current transaction was started (null if no transaction is active). If the current query is the first of its transaction, the value of this column is the same as that of the **query_start** column.
query_start	timestampwith time zone	Time when the currently active query was started, or time when the last query was started if **state** is not **active**
state_change	timestampwith time zone	Time when the **state** was last changed
waiting	boolean	Whether the backend is currently waiting on a lock. If yes, the value is **true**.
enqueue	text	Unsupported currently

Name	Type	Description

state	text	Overall status of this backend. The value must be one of the following:
		● **active**: The backend is executing a query.
		● **idle**: The backend is waiting for a new client command.
		● **idle in transaction**: The backend is in a transaction, but is not currently executing a query.
		● **idle in transaction (aborted)**: This state is similar to **idle in transaction**, except that one of the statements in the transaction caused an error.
		● **fastpath function call**: The backend is executing a fast-path function.
		● **disabled**: This state is reported if **track_activities** is disabled in this backend.
		NOTE
		Common users can view their own session status only. The state information of other accounts is empty. For example, after the **judy** user is connected to the database, the state information of the **joe** user and the initial user **omm** in **pg_stat_activity** is empty.
		`openGauss=# SELECT datname, usename, usesysid,state,pid FROM pg_stat_activity;`
		`datname    | usename | usesysid | state    |       pid`
		`----------+---------+----------+--------`
		`+-----------------postgres  | omm      | 10`
		`|       |139968752121616`
		`postgres  | omm      | 10       |        |`
		`139968903116560`
		`db_tpcds  | judy     | 16398    | active |`
		`139968391403280`
		`postgres  | omm      | 10       |        |`
		`139968643069712`
		`postgres  | omm      | 10       |        |`
		`139968680818448`
		`postgres  | joe      | 16390    |        |`
		`139968563377936`
		`(6 rows)`

resource_pool	name	Resource pool used by the user
query_id	bigint	ID of a query
query	text	Latest query at the backend. If **state** is **active**, this column shows the ongoing query. In all other states, it shows the last query that was executed.
unique_sql_id	bigint	Unique SQL statement ID.

Name	Type	Description
trace_id	text	Driver-specific trace ID, which is associated with an application request.

14.1.7.11 THREAD_WAIT_STATUS

THREAD_WAIT_STATUS allows you to test the block waiting status of the backend thread and auxiliary thread in the current instance. For details about events, see **Table 13-297**, **Table 13-298**, **Table 13-299**, and **Table 13-300**.

Table 14-84 THREAD_WAIT_STATUS columns

Name	Type	Description
node_name	text	Database process name
db_name	text	Database name
thread_name	text	Thread name
query_id	bigint	Query ID. The value of this column is the same as that of **debug_query_id**.
tid	bigint	Thread ID of the current thread
sessionid	bigint	Session ID
lwtid	integer	Lightweight thread ID of the current thread
psessionid	bigint	Parent thread of the streaming thread
tlevel	integer	Level of the streaming thread
smpid	integer	Concurrent thread ID
wait_status	text	Waiting status of the current thread. For details about the waiting status, see **Table 13-297**.
wait_event	text	If **wait_status** is **acquire lock**, **acquire lwlock**, or **wait io**, this column describes the lock, lightweight lock, and I/O information, respectively. If **wait_status** is not any of the three values, this column is empty.
locktag	text	Information about the lock that the current thread is waiting for
lockmode	text	Lock mode that the current thread is waiting to obtain The values include modes in the table-level lock, row-level lock, and page-level lock.
block_sessionid	bigint	ID of the session that blocks the current thread from obtaining the lock

Name	Type	Description
global_sessionid	text	Global session ID

14.1.7.12 GLOBAL_THREAD_WAIT_STATUS

GLOBAL_THREAD_WAIT_STATUS allows you to test the block waiting status of backend threads and auxiliary threads on all nodes. To query this view, you must have the **monadmin** permission. For details about events, see **Table 13-297**, **Table 13-298**, **Table 13-299**, and **Table 13-300**.

In **GLOBAL_THREAD_WAIT_STATUS**, you can see all the call hierarchy relationships between threads of the SQL statements on all nodes in the database, and the block waiting status for each thread. With this view, you can easily locate the causes of process hang and similar issues.

The definitions of **GLOBAL_THREAD_WAIT_STATUS** and **THREAD_WAIT_STATUS** are the same, because the **GLOBAL_THREAD_WAIT_STATUS** view is essentially the query summary of the **THREAD_WAIT_STATUS** view on each node in the database.

Table 14-85 GLOBAL_THREAD_WAIT_STATUS columns

Name	Type	Description
node_name	text	Database process name
db_name	text	Database name
thread_name	text	Thread name
query_id	bigint	Query ID. The value of this column is the same as that of **debug_query_id**.
tid	bigint	Thread ID of the current thread
sessionid	bigint	Session ID
lwtid	integer	Lightweight thread ID of the current thread
psessionid	bigint	Parent thread of the streaming thread
tlevel	integer	Level of the streaming thread
smpid	integer	Concurrent thread ID
wait_status	text	Waiting status of the current thread. For details about the waiting status, see **Table 13-297**.

Name	Type	Description
wait_event	text	If **wait_status** is **acquire lock**, **acquire lwlock**, or **wait io**, this column describes the lock, lightweight lock, and I/O information, respectively. If **wait_status** is not any of the three values, this column is empty.
locktag	text	Information about the lock that the current thread is waiting for
lockmode	text	Lock mode that the current thread is waiting to obtain The values include modes in the table-level lock, row-level lock, and page-level lock.
block_sessionid	bigint	ID of the session that blocks the current thread from obtaining the lock
global_sessionid	text	Global session ID

14.1.7.13 LOCAL_THREADPOOL_STATUS

LOCAL_THREADPOOL_STATUS displays the status of worker threads and sessions in a thread pool. This view is valid only when **enable_thread_pool** is set to **on**.

Table 14-86 LOCAL_THREADPOOL_STATUS columns

Name	Type	Description
node_name	text	Database process name
group_id	integer	ID of the thread pool group
bind_numa_id	integer	NUMA ID to which the thread pool group is bound
bind_cpu_number	integer	Information about the CPU to which the thread pool group is bound. If no CPUs are bound, the value is **NULL**.
listener	integer	Number of listener threads in the thread pool group

Name	Type	Description
worker_info	text	Information about threads in the thread pool, including: ● **default**: Number of initial threads in the thread pool group ● **new**: Number of new threads in the thread pool group ● **expect**: Expected number of threads in the thread pool group ● **actual**: Actual number of threads in the thread pool group ● **idle**: Number of idle threads in the thread pool group ● **pending**: Number of pending threads in the thread pool group
session_info	text	Information about sessions in the thread pool, including: ● **total**: Total number of sessions in the thread pool group ● **waiting**: Number of sessions pending scheduling in the thread pool group ● **running**: Number of running sessions in the thread pool group ● **idle**: Number of idle sessions in the thread pool group
stream_info	text	Stream pool information, including: ● **total**: total number of threads in the stream pool group. ● **running**: number of threads that are being executed in the stream pool. ● **idle**: number of idle threads in the stream pool.

14.1.7.14 GLOBAL_THREADPOOL_STATUS

GLOBAL_THREADPOOL_STATUS displays the status of worker threads and sessions in thread pools on all nodes. Columns in this view are the same as those in **Table 14-86**.

14.1.7.15 SESSION_CPU_RUNTIME

SESSION_CPU_RUNTIME displays load management information about CPU usage of ongoing complex jobs executed by the current user. The current feature is a lab feature. Contact Huawei technical support before using it.

Table 14-87 SESSION_CPU_RUNTIME columns

Name	Type	Description
datid	oid	OID of the database that this backend is connected to
usename	name	Name of the user logged in to the backend
pid	bigint	Thread ID of the backend
start_time	timestamp with time zone	Time when the statement starts to be executed
min_cpu_time	bigint	Minimum CPU time of the statement across the database nodes, in ms
max_cpu_time	bigint	Maximum CPU time of the statement across the database nodes, in ms
total_cpu_time	bigint	Total CPU time of the statement across the database nodes, in ms
query	text	Statement being executed
node_group	text	Logical database of the user running the statement
top_cpu_dn	text	Top N CPU usage

14.1.7.16 SESSION_MEMORY_RUNTIME

SESSION_MEMORY_RUNTIME displays load management information about memory usage of ongoing complex jobs executed by the current user. The current feature is a lab feature. Contact Huawei technical support before using it.

Table 14-88 SESSION_MEMORY_RUNTIME columns

Name	Type	Description
datid	oid	OID of the database that this backend is connected to
usename	name	Name of the user logged in to the backend
pid	bigint	Thread ID of the backend
start_time	timestamp with time zone	Time when the statement starts to be executed
min_peak_memory	integer	Minimum memory peak of the statement across the database nodes, in MB

Name	Type	Description
max_peak_memory	integer	Maximum memory peak of the statement across the database nodes, in MB
spill_info	text	Information about statement spill to the database nodes ● **None**: The statement has not been spilled to disks on the database nodes. ● **All**: The statement has been spilled to disks on the database nodes. ● *[a:b]*: The statement has been spilled to disks on *a* of *b* database nodes.
query	text	Statement being executed
node_group	text	Logical database of the user running the statement
top_mem_dn	text	Top N memory usage

14.1.7.17 STATEMENT_IOSTAT_COMPLEX_RUNTIME

STATEMENT_IOSTAT_COMPLEX_RUNTIME displays I/O load management information about ongoing jobs executed by the current user. The current feature is a lab feature. Contact Huawei technical support before using it. IOPS is counted by ones for column storage and by 10 thousands for row storage.

Table 14-89 STATEMENT_IOSTAT_COMPLEX_RUNTIME columns

Name	Type	Description
query_id	bigint	Job ID
mincurriops	integer	Minimum I/O of the job across database nodes
maxcurriops	integer	Maximum I/O of the job across database nodes
minpeakiops	integer	Minimum peak I/O of the current job across database nodes
maxpeakiops	integer	Maximum peak I/O of the current job across database nodes
io_limits	integer	**io_limits** set for the job
io_priority	text	**io_priority** set for the job
query	text	Job
node_group	text	Logical database of the user running the job

Name	Type	Description
curr_io_limits	integer	Real-time **io_limits** value when **io_priority** is used to control I/Os

14.1.7.18 LOCAL_ACTIVE_SESSION

LOCAL_ACTIVE_SESSION displays samples in the **ACTIVE SESSION PROFILE** memory on the current node.

Table 14-90 LOCAL_ACTIVE_SESSION columns

Name	Type	Description
sampleid	bigint	Sample ID
sample_time	timestamp with time zone	Sampling time
need_flush_sample	boolean	Specifies whether the sample needs to be flushed to disks.
databaseid	oid	Database ID
thread_id	bigint	Thread ID
sessionid	bigint	Session ID
start_time	timestamp with time zone	Start time of a session
event	text	Event name
lwtid	integer	Lightweight thread ID of the current thread
psessionid	bigint	Parent thread of the streaming thread
tlevel	integer	Level of the streaming thread, which corresponds to the level (**id**) of the execution plan
smpid	integer	Concurrent thread ID in SMP execution mode
userid	oid	ID of a session user
application_name	text	Name of an application
client_addr	inet	IP address of a client
client_hostname	text	Name of a client

Name	Type	Description
client_port	integer	TCP port number used by a client to communicate with the backend
query_id	bigint	debug query id
unique_query_id	bigint	unique query id
user_id	oid	User ID in the key of the unique query
cn_id	integer	CN ID. On a DN, this parameter indicates the ID of the node that delivers the unique SQL statement, that is, the value of **cn_id** in the key of the unique query.
unique_query	text	Standardized text string of the unique SQL statement
locktag	text	Information of a lock that the session waits for, which can be parsed using **locktag_decode**
lockmode	text	Mode of a lock that the session waits for
block_sessionid	bigint	Blocks a session from obtaining the session ID of a lock if the session is waiting for the lock.
final_block_sessionid	bigint	ID of the blocked session at the source end
wait_status	text	Provides more details about an event column.
global_sessionid	text	Global session ID
xact_start_time	timestamp with time zone	Start time of the transaction
query_start_time	timestamp with time zone	Time when the statement starts to be executed
state	text	Current statement state The value can be **active**, **idle in transaction**, **fastpath function call**, **idle in transaction (aborted)**, **disabled**, or **retrying**.

14.1.8 Transaction

14.1.8.1 TRANSACTIONS_PREPARED_XACTS

TRANSACTIONS_PREPARED_XACTS displays information about transactions that are currently prepared for two-phase commit.

Table 14-91 TRANSACTIONS_PREPARED_XACTS columns

Name	Type	Description
transaction	xid	Numeric transaction identifier of the prepared transaction
gid	text	Global transaction identifier that was assigned to the transaction
prepared	timestamp with time zone	Time at which the transaction is prepared for commit
owner	name	Name of the user that executes the transaction
database	name	Name of the database in which the transaction is executed

14.1.8.2 SUMMARY_TRANSACTIONS_PREPARED_XACTS

SUMMARY_TRANSACTIONS_PREPARED_XACTS displays information about transactions that are currently prepared for two-phase commit on the primary database node in the database.

Table 14-92 SUMMARY_TRANSACTIONS_PREPARED_XACTS columns

Name	Type	Description
transaction	xid	Numeric transaction identifier of the prepared transaction
gid	text	Global transaction identifier that was assigned to the transaction
prepared	timestamp with time zone	Time at which the transaction is prepared for commit
owner	name	Name of the user that executes the transaction
database	name	Name of the database in which the transaction is executed

14.1.8.3 GLOBAL_TRANSACTIONS_PREPARED_XACTS

GLOBAL_TRANSACTIONS_PREPARED_XACTS displays information about transactions that are currently prepared for two-phase commit on each node.

Table 14-93 GLOBAL_TRANSACTIONS_PREPARED_XACTS columns

Name	Type	Description
transaction	xid	Numeric transaction identifier of the prepared transaction
gid	text	Global transaction identifier that was assigned to the transaction
prepared	timestamp with time zone	Time at which the transaction is prepared for commit
owner	name	Name of the user that executes the transaction
database	name	Name of the database in which the transaction is executed

14.1.8.4 TRANSACTIONS_RUNNING_XACTS

TRANSACTIONS_RUNNING_XACTS displays information about running transactions on the current node.

Table 14-94 TRANSACTIONS_RUNNING_XACTS columns

Name	Type	Description
handle	integer	Slot handle in the transaction manager corresponding to a transaction. The value is fixed at **–1**.
gxid	xid	Transaction ID
state	tinyint	Transaction status (**3**: prepared or **0**: starting)
node	text	Node name
xmin	xid	Minimum transaction ID on the node
vacuum	boolean	Whether the current transaction is lazy vacuum
timeline	bigint	Number of database restarts
prepare_xid	xid	Transaction ID in the **prepared** state. If the state is not **prepared**, the value is **0**.
pid	bigint	Thread ID corresponding to the transaction
next_xid	xid	Transaction ID sent by other nodes to the current node. The value is fixed at **0**.

14.1.8.5 SUMMARY_TRANSACTIONS_RUNNING_XACTS

SUMMARY_TRANSACTIONS_RUNNING_XACTS displays information about the running transactions on each node in the cluster. The column content is the same as that of **TRANSACTIONS_RUNNING_XACTS**.

Table 14-95 SUMMARY_TRANSACTIONS_RUNNING_XACTS columns

Name	Type	Description
handle	integer	Slot handle in the transaction manager corresponding to a transaction. The value is fixed at **–1**.
gxid	xid	Transaction ID
state	tinyint	Transaction status (**3**: prepared or **0**: starting)
node	text	Node name
xmin	xid	Minimum transaction ID on the node
vacuum	boolean	Whether the current transaction is lazy vacuum
timeline	bigint	Number of database restarts
prepare_xid	xid	Transaction ID in the **prepared** state. If the state is not **prepared**, the value is **0**.
pid	bigint	Thread ID corresponding to the transaction
next_xid	xid	Transaction ID sent by other nodes to the current node. The value is fixed at **0**.

14.1.8.6 GLOBAL_TRANSACTIONS_RUNNING_XACTS

GLOBAL_TRANSACTIONS_RUNNING_XACTS displays information about the running transactions on each node in the cluster.

Table 14-96 GLOBAL_TRANSACTIONS_RUNNING_XACTS columns

Name	Type	Description
handle	integer	Slot handle in the transaction manager corresponding to a transaction. The value is fixed at **–1**.
gxid	xid	Transaction ID
state	tinyint	Transaction status (**3**: prepared or **0**: starting)
node	text	Node name
xmin	xid	Minimum transaction ID on the node

Name	Type	Description
vacuum	boolean	Whether the current transaction is lazy vacuum
timeline	bigint	Number of database restarts
prepare_xid	xid	Transaction ID in the **prepared** state. If the state is not **prepared**, the value is **0**.
pid	bigint	Thread ID corresponding to the transaction
next_xid	xid	Transaction ID sent by other nodes to the current node. The value is fixed at **0**.

14.1.9 Query

14.1.9.1 STATEMENT

STATEMENT obtains information about executed statements (normalized SQL statements) on the current node. To query a view, you must have the sysadmin or monitor admin permission. You can view all statistics about normalized SQL statements received by the primary database node and other database nodes, whereas you can view only the statistics about normalized SQL statements executed on other database nodes.

Table 14-97 STATEMENT columns

Name	Type	Description
node_name	name	Database process name
node_id	integer	ID of a node
user_name	name	Username
user_id	oid	OID of the user
unique_sql_id	bigint	ID of the normalized SQL statement
query	text	Normalized SQL statement Note: The length is controlled by **track_activity_query_size**.
n_calls	bigint	Number of calls
min_elapse_time	bigint	Minimum execution time of the SQL statement in the kernel (unit: μs)
max_elapse_time	bigint	Maximum execution time of the SQL statement in the kernel (unit: μs)
total_elapse_time	bigint	Total execution time of the SQL statement in the kernel (unit: μs)

Name	Type	Description
n_returned_rows	bigint	Number of rows in the result set returned by the **SELECT** statement
n_tuples_fetched	bigint	Number of rows randomly scanned
n_tuples_returned	bigint	Number of rows sequentially scanned
n_tuples_inserted	bigint	Number of rows inserted
n_tuples_updated	bigint	Number of rows updated
n_tuples_deleted	bigint	Number of rows deleted
n_blocks_fetched	bigint	Number of buffer block access times
n_blocks_hit	bigint	Number of buffer block hits
n_soft_parse	bigint	Number of soft parsing times. The value of **n_soft_parse** plus the value of **n_hard_parse** may be greater than the value of **n_calls** because the number of subqueries is not counted in the value of **n_calls**.
n_hard_parse	bigint	Number of hard parsing times. The value of **n_soft_parse** plus the value of **n_hard_parse** may be greater than the value of **n_calls** because the number of subqueries is not counted in the value of **n_calls**.
db_time	bigint	Valid DB time, which is accumulated if multiple threads are involved (unit: μs)
cpu_time	bigint	CPU time (unit: μs)
execution_time	bigint	Execution time in the executor (unit: μs)
parse_time	bigint	SQL parsing time (unit: μs)
plan_time	bigint	SQL plan generation time (unit: μs)
rewrite_time	bigint	SQL rewriting time (unit: μs)
pl_execution_time	bigint	Execution time of PL/pgSQL (unit: μs)
pl_compilation_time	bigint	Compilation time of PL/pgSQL (unit: μs)
data_io_time	bigint	I/O time (unit: μs)

Name	Type	Description
net_send_info	text	Network status of messages sent through a physical connection, including the time (unit: µs), number of calls, and throughput (unit: byte). This field can be used to analyze the network overhead of SQL in a distributed system. This field is not supported in standalone mode. Example: {"time":xxx, "n_calls":xxx, "size":xxx}.
net_recv_info	text	Network status of messages received through a physical connection, including the time (unit: µs), number of calls, and throughput (unit: byte). This field can be used to analyze the network overhead of SQL in a distributed system. This field is not supported in standalone mode. Example: {"time":xxx, "n_calls":xxx, "size":xxx}.
net_stream_send_info	text	Network status of messages sent through a logical connection, including the time (unit: µs), number of calls, and throughput (unit: byte). This field can be used to analyze the network overhead of SQL in a distributed system. This field is not supported in standalone mode. Example: {"time":xxx, "n_calls":xxx, "size":xxx}.
net_stream_recv_info	text	Network status of messages received through a logical connection, including the time (unit: µs), number of calls, and throughput (unit: byte). This field can be used to analyze the network overhead of SQL in a distributed system. This field is not supported in standalone mode. Example: {"time":xxx, "n_calls":xxx, "size":xxx}.
sort_count	bigint	Sorting count
sort_time	bigint	Sorting duration (unit: µs)
sort_mem_used	bigint	Size of work memory used during sorting (unit: KB)
sort_spill_count	bigint	Count of file writing when data is flushed to disks during sorting
sort_spill_size	bigint	File size used when data is flushed to disks during sorting (unit: KB)
hash_count	bigint	Hashing count
hash_time	bigint	Hashing duration (unit: µs)

Name	Type	Description
hash_mem_used	bigint	Size of work memory used during hashing (unit: KB)
hash_spill_count	bigint	Count of file writing when data is flushed to disks during hashing
hash_spill_size	bigint	File size used when data is flushed to disks during hashing (unit: KB)
last_updated	timestamp with time zone	Last time when the statement was updated

14.1.9.2 SUMMARY_STATEMENT

SUMMARY_STATEMENT obtains all information about executed statements (normalized SQL statements) on the primary database node and other database nodes.

Table 14-98 SUMMARY_STATEMENT columns

Name	Type	Description
node_name	name	Database process name
node_id	integer	ID of a node
user_name	name	Username
user_id	oid	OID of the user
unique_sql_id	bigint	ID of the normalized SQL statement
query	text	Normalized SQL statement Note: The length is controlled by **track_activity_query_size**.
n_calls	bigint	Number of calls
min_elapse_time	bigint	Minimum execution time of the SQL statement in the kernel (unit: μs)
max_elapse_time	bigint	Maximum execution time of the SQL statement in the kernel (unit: μs)
total_elapse_time	bigint	Total execution time of the SQL statement in the kernel (unit: μs)
n_returned_rows	bigint	Number of rows in the result set returned by the **SELECT** statement
n_tuples_fetched	bigint	Number of rows randomly scanned

Name	Type	Description
n_tuples_returned	bigint	Number of rows sequentially scanned
n_tuples_inserted	bigint	Number of rows inserted
n_tuples_updated	bigint	Number of rows updated
n_tuples_deleted	bigint	Number of rows deleted
n_blocks_fetched	bigint	Number of buffer block access times
n_blocks_hit	bigint	Number of buffer block hits
n_soft_parse	bigint	Number of soft parsing times
n_hard_parse	bigint	Number of hard parsing times
db_time	bigint	Valid DB time, which is accumulated if multiple threads are involved (unit: μs)
cpu_time	bigint	CPU time (unit: μs)
execution_time	bigint	Execution time in the executor (unit: μs)
parse_time	bigint	SQL parsing time (unit: μs)
plan_time	bigint	SQL plan generation time (unit: μs)
rewrite_time	bigint	SQL rewriting time (unit: μs)
pl_execution_time	bigint	Execution time of PL/pgSQL (unit: μs)
pl_compilation_time	bigint	Compilation time of PL/pgSQL (unit: μs)
data_io_time	bigint	I/O time (unit: μs)
net_send_info	text	Network status of messages sent through a physical connection, including the time (unit: μs), number of calls, and throughput (unit: byte). This field can be used to analyze the network overhead of SQL in a distributed system. This field is not supported in standalone mode. Example: {"time":xxx, "n_calls":xxx, "size":xxx}.
net_recv_info	text	Network status of messages received through a physical connection, including the time (unit: μs), number of calls, and throughput (unit: byte). This field can be used to analyze the network overhead of SQL in a distributed system. This field is not supported in standalone mode. Example: {"time":xxx, "n_calls":xxx, "size":xxx}.

Name	Type	Description
net_stream_send_info	text	Network status of messages sent through a logical connection, including the time (unit: μs), number of calls, and throughput (unit: byte). This field can be used to analyze the network overhead of SQL in a distributed system. This field is not supported in standalone mode. Example: {"time":xxx, "n_calls":xxx, "size":xxx}.
net_stream_recv_info	text	Network status of messages received through a logical connection, including the time (unit: μs), number of calls, and throughput (unit: byte). This field can be used to analyze the network overhead of SQL in a distributed system. This field is not supported in standalone mode. Example: {"time":xxx, "n_calls":xxx, "size":xxx}.
last_updated	timestamp with time zone	Last time when the statement was updated
sort_count	bigint	Sorting count
sort_time	bigint	Sorting duration (unit: μs)
sort_mem_used	bigint	Size of work memory used during sorting (unit: KB)
sort_spill_count	bigint	Count of file writing when data is flushed to disks during sorting
sort_spill_size	bigint	File size used when data is flushed to disks during sorting (unit: KB)
hash_count	bigint	Hashing count
hash_time	bigint	Hashing duration (unit: μs)
hash_mem_used	bigint	Size of work memory used during hashing (unit: KB)
hash_spill_count	bigint	Count of file writing when data is flushed to disks during hashing
hash_spill_size	bigint	File size used when data is flushed to disks during hashing (unit: KB)

14.1.9.3 STATEMENT_COUNT

STATEMENT_COUNT displays statistics about five types of running statements (**SELECT**, **INSERT**, **UPDATE**, **DELETE**, and **MERGE INTO**) as well as DDL, DML, and DCL statements on the current node of the database.

NOTE

By querying the **STATEMENT_COUNT** view, a common user can view statistics only about this user on the current node, whereas an administrator can view statistics about all users on the current node. When the database or the node is restarted, the statistics are cleared and the counting restarts. The system counts when a node receives a query, including a query inside the database. For example, when the primary database node receives a query and distributes multiple queries to other database nodes, the queries are counted accordingly on the database nodes.

Table 14-99 STATEMENT_COUNT columns

Name	Type	Description
node_name	text	Database process name
user_name	text	Username
select_count	bigint	Statistical result of the **SELECT** statement
update_count	bigint	Statistical result of the **UPDATE** statement
insert_count	bigint	Statistical result of the **INSERT** statement
delete_count	bigint	Statistical result of the **DELETE** statement
mergeinto_count	bigint	Statistical result of the **MERGE INTO** statement
ddl_count	bigint	Number of DDL statements
dml_count	bigint	Number of DML statements
dcl_count	bigint	Number of DCL statements
total_select_elapse	bigint	Total response time of **SELECT** statements (unit: μs)
avg_select_elapse	bigint	Average response time of **SELECT** statements (unit: μs)
max_select_elapse	bigint	Maximum response time of **SELECT** statements (unit: μs)
min_select_elapse	bigint	Minimum response time of **SELECT** statements (unit: μs)
total_update_elapse	bigint	Total response time of **UPDATE** statements (unit: μs)
avg_update_elapse	bigint	Average response time of **UPDATE** statements (unit: μs)
max_update_elapse	bigint	Maximum response time of **UPDATE** statements (unit: μs)
min_update_elapse	bigint	Minimum response time of **UPDATE** statements (unit: μs)

Name	Type	Description
total_insert_elapse	bigint	Total response time of **INSERT** statements (unit: μs)
avg_insert_elapse	bigint	Average response time of **INSERT** statements (unit: μs)
max_insert_elapse	bigint	Maximum response time of **INSERT** statements (unit: μs)
min_insert_elapse	bigint	Minimum response time of **INSERT** statements (unit: μs)
total_delete_elapse	bigint	Total response time of **DELETE** statements (unit: μs)
avg_delete_elapse	bigint	Average response time of **DELETE** statements (unit: μs)
max_delete_elapse	bigint	Maximum response time of **DELETE** statements (unit: μs)
min_delete_elapse	bigint	Minimum response time of **DELETE** statements (unit: μs)

14.1.9.4 GLOBAL_STATEMENT_COUNT

GLOBAL_STATEMENT_COUNT displays statistics about five types of running statements (**SELECT**, **INSERT**, **UPDATE**, **DELETE**, and **MERGE INTO**) as well as DDL, DML, and DCL statements on each node of the database.

Table 14-100 GLOBAL_STATEMENT_COUNT columns

Name	Type	Description
node_name	text	Node name
user_name	text	Username
select_count	bigint	Statistical result of the **SELECT** statement
update_count	bigint	Statistical result of the **UPDATE** statement
insert_count	bigint	Statistical result of the **INSERT** statement
delete_count	bigint	Statistical result of the **DELETE** statement
mergeinto_count	bigint	Statistical result of the **MERGE INTO** statement
ddl_count	bigint	Number of DDL statements
dml_count	bigint	Number of DML statements

Name	Type	Description
dcl_count	bigint	Number of DCL statements
total_select_elapse	bigint	Total response time of **SELECT** statements (unit: μs)
avg_select_elapse	bigint	Average response time of **SELECT** statements (unit: μs)
max_select_elapse	bigint	Maximum response time of **SELECT** statements (unit: μs)
min_select_elapse	bigint	Minimum response time of **SELECT** statements (unit: μs)
total_update_elapse	bigint	Total response time of **UPDATE** statements (unit: μs)
avg_update_elapse	bigint	Average response time of **UPDATE** statements (unit: μs)
max_update_elapse	bigint	Maximum response time of **UPDATE** statements (unit: μs)
min_update_elapse	bigint	Minimum response time of **UPDATE** statements (unit: μs)
total_insert_elapse	bigint	Total response time of **INSERT** statements (unit: μs)
avg_insert_elapse	bigint	Average response time of **INSERT** statements (unit: μs)
max_insert_elapse	bigint	Maximum response time of **INSERT** statements (unit: μs)
min_insert_elapse	bigint	Minimum response time of **INSERT** statements (unit: μs)
total_delete_elapse	bigint	Total response time of **DELETE** statements (unit: μs)
avg_delete_elapse	bigint	Average response time of **DELETE** statements (unit: μs)
max_delete_elapse	bigint	Maximum response time of **DELETE** statements (unit: μs)
min_delete_elapse	bigint	Minimum response time of **DELETE** statements (unit: μs)

14.1.9.5 SUMMARY_STATEMENT_COUNT

SUMMARY_STATEMENT_COUNT displays statistics about five types of running statements (**SELECT**, **INSERT**, **UPDATE**, **DELETE**, and **MERGE INTO**) as well as DDL, DML, and DCL statements on all nodes (database nodes) of the database.

Table 14-101 SUMMARY_STATEMENT_COUNT columns

Name	Type	Description
user_name	text	Username
select_count	numeric	Statistical result of the **SELECT** statement
update_count	numeric	Statistical result of the **UPDATE** statement
insert_count	numeric	Statistical result of the **INSERT** statement
delete_count	numeric	Statistical result of the **DELETE** statement
mergeinto_count	numeric	Statistical result of the **MERGE INTO** statement
ddl_count	numeric	Number of DDL statements
dml_count	numeric	Number of DML statements
dcl_count	numeric	Number of DCL statements
total_select_elapse	numeric	Total response time of **SELECT** statements (unit: μs)
avg_select_elapse	bigint	Average response time of **SELECT** statements (unit: μs)
max_select_elapse	bigint	Maximum response time of **SELECT** statements (unit: μs)
min_select_elapse	bigint	Minimum response time of **SELECT** statements (unit: μs)
total_update_elapse	numeric	Total response time of **UPDATE** statements (unit: μs)
avg_update_elapse	bigint	Average response time of **UPDATE** statements (unit: μs)
max_update_elapse	bigint	Maximum response time of **UPDATE** statements (unit: μs)
min_update_elapse	bigint	Minimum response time of **UPDATE** statements (unit: μs)
total_insert_elapse	numeric	Total response time of **INSERT** statements (unit: μs)
avg_insert_elapse	bigint	Average response time of **INSERT** statements (unit: μs)
max_insert_elapse	bigint	Maximum response time of **INSERT** statements (unit: μs)

Name	Type	Description
min_insert_elapse	bigint	Minimum response time of **INSERT** statements (unit: μs)
total_delete_elapse	numeric	Total response time of **DELETE** statements (unit: μs)
avg_delete_elapse	bigint	Average response time of **DELETE** statements (unit: μs)
max_delete_elapse	bigint	Maximum response time of **DELETE** statements (unit: μs)
min_delete_elapse	bigint	Minimum response time of **DELETE** statements (unit: μs)

14.1.9.6 GLOBAL_STATEMENT_COMPLEX_HISTORY

GLOBAL_STATEMENT_COMPLEX_HISTORY displays load management information about completed jobs executed on each node. The current feature is a lab feature. Contact Huawei technical support before using it.

Table 14-102 GLOBAL_STATEMENT_COMPLEX_HISTORY columns

Name	Type	Description
datid	oid	OID of the database that the backend is connected to
dbname	text	Name of the database that the backend is connected to
schemaname	text	Schema name
nodename	text	Node name
username	text	Username used for connecting to the backend
application_name	text	Name of the application connected to the backend
client_addr	inet	IP address of the client connected to the backend. If this column is null, it indicates either the client is connected via a Unix socket on the server or this is an internal process, such as autovacuum.

Name	Type	Description
client_hostname	text	Host name of the connected client, as reported by a reverse DNS lookup of **client_addr**. This column will be non-null only for IP connections and only when **log_hostname** is enabled.
client_port	integer	TCP port number that the client uses for communication with this backend (**-1** if a Unix socket is used)
query_band	text	Job type, which is specified by the GUC parameter **query_band**. The default value is a null string.
block_time	bigint	Duration that the statement is blocked before being executed, including the statement parsing and optimization duration (unit: ms)
start_time	timestamp with time zone	Time when the statement starts to be executed
finish_time	timestamp with time zone	Time when the statement execution ends
duration	bigint	Execution time of the statement (unit: ms)
estimate_total_time	bigint	Estimated execution time of the statement (unit: ms)
status	text	Final statement execution status, which can be **finished** (normal) or **aborted** (abnormal).
abort_info	text	Exception information displayed if the final statement execution status is **aborted**
resource_pool	text	Resource pool used by the user
control_group	text	Cgroup used by the statement
estimate_memory	integer	Estimated memory used by the statement
min_peak_memory	integer	Minimum memory peak of the statement across the database nodes (unit: MB)
max_peak_memory	integer	Maximum memory peak of the statement across the database nodes (unit: MB)

Name	Type	Description
average_peak_memory	integer	Average memory usage during statement execution (unit: MB)
memory_skew_percent	integer	Memory usage skew of the statement among the database nodes
spill_info	text	Information about statement spill to the database nodes ● **None**: The statement has not been spilled to disks on the database nodes. ● **All**: The statement has been spilled to disks on the database nodes. ● *[a:b]*: The statement has been spilled to disks on *a* of *b* database nodes.
min_spill_size	integer	Minimum spilled data among database nodes when a spill occurs (unit: MB) (default value: **0**)
max_spill_size	integer	Maximum spilled data among database nodes when a spill occurs (unit: MB) (default value: **0**)
average_spill_size	integer	Average spilled data among database nodes when a spill occurs (unit: MB) (default value: **0**)
spill_skew_percent	integer	Database node spill skew when a spill occurs
min_dn_time	bigint	Minimum execution time of the statement across the database nodes (unit: ms)
max_dn_time	bigint	Maximum execution time of the statement across the database nodes (unit: ms)
average_dn_time	bigint	Average execution time of the statement across the database nodes (unit: ms)
dntime_skew_percent	integer	Execution time skew of the statement among the database nodes
min_cpu_time	bigint	Minimum CPU time of the statement across the database nodes (unit: ms)
max_cpu_time	bigint	Maximum CPU time of the statement across the database nodes (unit: ms)

Name	Type	Description
total_cpu_time	bigint	Total CPU time of the statement across the database nodes (unit: ms)
cpu_skew_percent	integer	CPU time skew of the statement among the database nodes
min_peak_iops	integer	Minimum IOPS peak of the statement across the database nodes. It is counted by ones in a column-store table and by ten thousands in a row-store table.
max_peak_iops	integer	Maximum IOPS peak of the statement across the database nodes. It is counted by ones in a column-store table and by ten thousands in a row-store table.
average_peak_iops	integer	Average IOPS peak of the statement across the database nodes. It is counted by ones in a column-store table and by ten thousands in a row-store table.
iops_skew_percent	integer	I/O skew of the statement among the database nodes
warning	text	Warning. The following warnings are displayed: ● Spill file size large than 256MB ● Broadcast size large than 100MB ● Early spill ● Spill times is greater than 3 ● Spill on memory adaptive ● Hash table conflict
queryid	bigint	Internal query ID used for statement execution
query	text	Statement executed
query_plan	text	Execution plan of the statement
node_group	text	Logical database of the user running the statement
cpu_top1_node_name	text	Name of the node with the highest CPU usage
cpu_top2_node_name	text	Name of the node with the second highest CPU usage
cpu_top3_node_name	text	Name of the node with the third highest CPU usage

Name	Type	Description
cpu_top4_node_name	text	Name of the node with the fourth highest CPU usage
cpu_top5_node_name	text	Name of the node with the fifth highest CPU usage
mem_top1_node_name	text	Name of the node with the highest memory usage
mem_top2_node_name	text	Name of the node with the second highest memory usage
mem_top3_node_name	text	Name of the node with the third highest memory usage
mem_top4_node_name	text	Name of the node with the fourth highest memory usage
mem_top5_node_name	text	Name of the node with the fifth highest memory usage
cpu_top1_value	bigint	CPU usage
cpu_top2_value	bigint	CPU usage
cpu_top3_value	bigint	CPU usage
cpu_top4_value	bigint	CPU usage
cpu_top5_value	bigint	CPU usage
mem_top1_value	bigint	Memory usage
mem_top2_value	bigint	Memory usage
mem_top3_value	bigint	Memory usage
mem_top4_value	bigint	Memory usage
mem_top5_value	bigint	Memory usage
top_mem_dn	text	Top N memory usage
top_cpu_dn	text	Top N CPU usage

14.1.9.7 GLOBAL_STATEMENT_COMPLEX_HISTORY_TABLE

GLOBAL_STATEMENT_COMPLEX_HISTORY_TABLE displays load management information about completed jobs executed on each node. The current feature is a lab feature. Contact Huawei technical support before using it. Data is dumped from the kernel to this system catalog. For details about the columns, see **GLOBAL_STATEMENT_COMPLEX_HISTORY**.

14.1.9.8 GLOBAL_STATEMENT_COMPLEX_RUNTIME

GLOBAL_STATEMENT_COMPLEX_RUNTIME displays load management records of jobs that are being executed by the current user on each node. The current feature is a lab feature. Contact Huawei technical support before using it.

Table 14-103 GLOBAL_STATEMENT_COMPLEX_RUNTIME columns

Name	Type	Description
datid	oid	OID of the database that the backend is connected to
dbname	name	Name of the database that the backend is connected to
schemaname	text	Schema name
nodename	text	Node name
username	name	Username used for connecting to the backend
application_name	text	Name of the application connected to the backend
client_addr	inet	IP address of the client connected to the backend. If this column is null, it indicates either the client is connected via a Unix socket on the server or this is an internal process, such as autovacuum.
client_hostname	text	Host name of the connected client, as reported by a reverse DNS lookup of **client_addr**. This column will be non-null only for IP connections and only when **log_hostname** is enabled.
client_port	integer	TCP port number that the client uses for communication with this backend (**-1** if a Unix socket is used)
query_band	text	Job type, which is specified by the GUC parameter **query_band**. The default value is a null string.
pid	bigint	Thread ID of the backend
block_time	bigint	Block time before the statement is executed (unit: ms)
start_time	timestamp with time zone	Time when the statement starts to be executed

Name	Type	Description
duration	bigint	Duration that a statement has been executed (unit: ms)
estimate_total_time	bigint	Estimated execution time of the statement (unit: ms)
estimate_left_time	bigint	Estimated remaining execution time of the statement (unit: ms)
enqueue	text	Resource status in workload management. The current feature is a lab feature. Contact Huawei technical support before using it.
resource_pool	name	Resource pool used by the user
control_group	text	Cgroup used by the statement
estimate_memory	integer	Estimated memory used by the statement (unit: MB).
min_peak_memory	integer	Minimum memory peak of the statement across the database nodes (unit: MB)
max_peak_memory	integer	Maximum memory peak of the statement across the database nodes (unit: MB)
average_peak_memory	integer	Average memory usage during statement execution (unit: MB)
memory_skew_percent	integer	Memory usage skew of the statement among the database nodes
spill_info	text	Information about statement spill to the database nodes ● **None**: The statement has not been spilled to disks on the database nodes. ● **All**: The statement has been spilled to disks on the database nodes. ● *[a:b]*: The statement has been spilled to disks on *a* of *b* database nodes.
min_spill_size	integer	Minimum spilled data among database nodes when a spill occurs (unit: MB) (default value: **0**)
max_spill_size	integer	Maximum spilled data among database nodes when a spill occurs (unit: MB) (default value: **0**)

Name	Type	Description
average_spill_size	integer	Average spilled data among database nodes when a spill occurs (unit: MB) (default value: **0**)
spill_skew_percent	integer	Database node spill skew when a spill occurs
min_dn_time	bigint	Minimum execution time of the statement across the database nodes (unit: ms)
max_dn_time	bigint	Maximum execution time of the statement across the database nodes (unit: ms)
average_dn_time	bigint	Average execution time of the statement across the database nodes (unit: ms)
dntime_skew_percen t	integer	Execution time skew of the statement among the database nodes
min_cpu_time	bigint	Minimum CPU time of the statement across the database nodes (unit: ms)
max_cpu_time	bigint	Maximum CPU time of the statement across the database nodes (unit: ms)
total_cpu_time	bigint	Total CPU time of the statement across the database nodes (unit: ms)
cpu_skew_percent	integer	CPU time skew of the statement among the database nodes
min_peak_iops	integer	Minimum IOPS peak of the statement across the database nodes. It is counted by ones in a column-store table and by ten thousands in a row-store table.
max_peak_iops	integer	Maximum IOPS peak of the statement across the database nodes. It is counted by ones in a column-store table and by ten thousands in a row-store table.
average_peak_iops	integer	Average IOPS peak of the statement across the database nodes. It is counted by ones in a column-store table and by ten thousands in a row-store table.
iops_skew_percent	integer	I/O skew of the statement among the database nodes

Name	Type	Description
warning	text	Warning. The following warnings are displayed: ● Spill file size large than 256MB ● Broadcast size large than 100MB ● Early spill ● Spill times is greater than 3 ● Spill on memory adaptive ● Hash table conflict
queryid	bigint	Internal query ID used for statement execution
query	text	Statement being executed
query_plan	text	Execution plan of the statement
node_group	text	Logical database of the user running the statement
top_cpu_dn	text	Top N CPU usage
top_mem_dn	text	Top N memory usage

14.1.9.9 STATEMENT_RESPONSETIME_PERCENTILE

STATEMENT_RESPONSETIME_PERCENTILE obtains the response times of 80% and 95% SQL statements in the database.

Table 14-104 STATEMENT_RESPONSETIME_PERCENTILE columns

Name	Type	Description
p80	bigint	Response time of 80% SQL statements in the database (unit: μs)
p95	bigint	Response time of 95% SQL statements in the database (unit: μs)

14.1.9.10 STATEMENT_COMPLEX_RUNTIME

STATEMENT_COMPLEX_RUNTIME displays load management information about jobs being executed by the current user on the primary database node. The current feature is a lab feature. Contact Huawei technical support before using it.

Table 14-105 STATEMENT_COMPLEX_RUNTIME columns

Name	Type	Description
datid	oid	OID of the database that the backend is connected to
dbname	name	Name of the database that the backend is connected to
schemaname	text	Schema name
nodename	text	Database process name
username	name	Username used for connecting to the backend
application_name	text	Name of the application connected to the backend
client_addr	inet	IP address of the client connected to the backend. If this column is null, it indicates either the client is connected via a Unix socket on the server or this is an internal process, such as autovacuum.
client_hostname	text	Host name of the connected client, as reported by a reverse DNS lookup of **client_addr**. This column will be non-null only for IP connections and only when **log_hostname** is enabled.
client_port	integer	TCP port number that the client uses for communication with this backend (**-1** if a Unix socket is used)
query_band	text	Job type, which is specified by the GUC parameter **query_band**. The default value is a null string.
pid	bigint	Thread ID of the backend
block_time	bigint	Block time before the statement is executed (unit: ms)
start_time	timestamp with time zone	Time when the statement starts to be executed
duration	bigint	Duration that a statement has been executed (unit: ms)
estimate_total_time	bigint	Estimated execution time of the statement (unit: ms)
estimate_left_time	bigint	Estimated remaining execution time of the statement (unit: ms)

Name	Type	Description
enqueue	text	Resource status in workload management. The current feature is a lab feature. Contact Huawei technical support before using it.
resource_pool	name	Resource pool used by the user
control_group	text	Cgroup used by the statement
estimate_memory	integer	Estimated memory used by the statement (unit: MB).
min_peak_memory	integer	Minimum memory peak of the statement across the database nodes (unit: MB)
max_peak_memory	integer	Maximum memory peak of the statement across the database nodes (unit: MB)
average_peak_memory	integer	Average memory usage during statement execution (unit: MB)
memory_skew_percent	integer	Memory usage skew of the statement among the database nodes
spill_info	text	Information about statement spill to the database nodes ● **None**: The statement has not been spilled to disks on the database nodes. ● **All**: The statement has been spilled to disks on the database nodes. ● *[a:b]*: The statement has been spilled to disks on *a* of *b* database nodes.
min_spill_size	integer	Minimum spilled data among database nodes when a spill occurs (unit: MB) (default value: **0**)
max_spill_size	integer	Maximum spilled data among database nodes when a spill occurs (unit: MB) (default value: **0**)
average_spill_size	integer	Average spilled data among database nodes when a spill occurs (unit: MB) (default value: **0**)
spill_skew_percent	integer	Database node spill skew when a spill occurs

Name	Type	Description
min_dn_time	bigint	Minimum execution time of the statement across the database nodes (unit: ms)
max_dn_time	bigint	Maximum execution time of the statement across the database nodes (unit: ms)
average_dn_time	bigint	Average execution time of the statement across the database nodes (unit: ms)
dntime_skew_percent	integer	Execution time skew of the statement among the database nodes
min_cpu_time	bigint	Minimum CPU time of the statement across the database nodes (unit: ms)
max_cpu_time	bigint	Maximum CPU time of the statement across the database nodes (unit: ms)
total_cpu_time	bigint	Total CPU time of the statement across the database nodes (unit: ms)
cpu_skew_percent	integer	CPU time skew of the statement among the database nodes
min_peak_iops	integer	Minimum IOPS peak of the statement across the database nodes. It is counted by ones in a column-store table and by ten thousands in a row-store table.
max_peak_iops	integer	Maximum IOPS peak of the statement across the database nodes. It is counted by ones in a column-store table and by ten thousands in a row-store table.
average_peak_iops	integer	Average IOPS peak of the statement across the database nodes. It is counted by ones in a column-store table and by ten thousands in a row-store table.
iops_skew_percent	integer	I/O skew of the statement among the database nodes

Name	Type	Description
warning	text	Warning. The following warnings are displayed: ● Spill file size large than 256MB ● Broadcast size large than 100MB ● Early spill ● Spill times is greater than 3 ● Spill on memory adaptive ● Hash table conflict
queryid	bigint	Internal query ID used for statement execution
query	text	Statement being executed
query_plan	text	Execution plan of the statement
node_group	text	Logical database of the user running the statement
top_cpu_dn	text	Top N CPU usage
top_mem_dn	text	Top N memory usage

14.1.9.11 STATEMENT_COMPLEX_HISTORY_TABLE

STATEMENT_COMPLEX_HISTORY_TABLE displays load management information about completed jobs executed on the current primary database node. (The current feature is a lab feature. Contact Huawei technical support before using it.) Data is dumped from the kernel to this system catalog. For details about columns, see **GS_WLM_SESSION_HISTORY**.

14.1.9.12 STATEMENT_COMPLEX_HISTORY

STATEMENT_COMPLEX_HISTORY displays load management information about a completed job executed on the primary database node. The current feature is a lab feature. Contact Huawei technical support before using it. For details about columns, see **GS_WLM_SESSION_HISTORY**.

14.1.9.13 STATEMENT_WLMSTAT_COMPLEX_RUNTIME

STATEMENT_WLMSTAT_COMPLEX_RUNTIME displays load management information about ongoing jobs executed by the current user. The current feature is a lab feature. Contact Huawei technical support before using it.

Table 14-106 STATEMENT_WLMSTAT_COMPLEX_RUNTIME columns

Name	Type	Description
datid	oid	OID of the database that the backend is connected to
datname	name	Name of the database that the backend is connected to
threadid	bigint	Thread ID of the backend
processid	integer	Process ID of the backend
usesysid	oid	OID of the user logged in to the backend
appname	text	Name of the application connected to the backend
usename	name	Name of the user logged in to the backend
priority	bigint	Priority of Cgroup where the statement is located
attribute	text	Attributes of the statement: ● **Ordinary**: default attribute of a statement before it is parsed by the database ● **Simple**: simple statement ● **Complicated**: complicated statement ● **Internal**: internal statement of the database
block_time	bigint	Pending duration of the statement by now (unit: s)
elapsed_time	bigint	Actual execution duration of the statement by now (unit: s)
total_cpu_time	bigint	Total CPU usage duration of the statement on the database nodes in the last period (unit: s)
cpu_skew_percent	integer	CPU usage skew percentage of the statement on the database nodes in the last period
statement_mem	integer	**statement_mem** used for executing the statement (reserved column)
active_points	integer	Number of concurrently active points occupied by the statement in the resource pool

Name	Type	Description
dop_value	integer	DOP value obtained by the statement from the resource pool
control_group	text	Unsupported currently
status	text	Unsupported currently
enqueue	text	Queuing status of the statement, including: ● **Global**: queuing in the global queue ● **Respool**: queuing in the resource pool queue ● **CentralQueue**: queuing on the CCN ● **Transaction**: being in a transaction block ● **StoredProc**: being in a stored procedure ● **None**: not in the queue ● **Forced None**: being forcibly executed (transaction block statement or stored procedure statement are) because the statement waiting time exceeds the specified value
resource_pool	name	Current resource pool where the statements are located
query	text	Latest query at the backend. If **state** is **active**, this column shows the ongoing query. In all other states, it shows the last query that was executed.
is_plana	boolean	Whether the execution of the statement occupies resources of other logical databases in logical database mode. The default value is **f** (does not occupy resources).
node_group	text	Logical database of the user running the statement

14.1.9.14 STATEMENT_HISTORY

STATEMENT_HISTORY displays information about execution statements on the current node. To query a view, you must have the **sysadmin** or **monitor admin** permission. The result can be queried only in the system database but cannot be queried in the user database.

Table 14-107 STATEMENT_HISTORY columns

Name	Type	Description
dbname	name	Database name.
schemaname	name	Schema name.
origin_node	integer	Node name.
user_name	name	Username.
application_name	text	Name of the application that sends a request.
client_addr	text	IP address of the client that sends a request.
client_port	integer	Port number of the client that sends a request.
unique_query_id	bigint	ID of the normalized SQL statement.
debug_query_id	bigint	ID of the unique SQL statement.
query	text	Normalized SQL statement.
start_time	timestamp with time zone	Time when a statement starts.
finish_time	timestamp with time zone	Time when a statement ends.
slow_sql_threshold	bigint	Standard for slow SQL statement execution.
transaction_id	bigint	Transaction ID.
thread_id	bigint	ID of an execution thread.
session_id	bigint	Session ID of a user.
n_soft_parse	bigint	Number of soft parsing times. The value of **n_soft_parse** plus the value of **n_hard_parse** may be greater than the value of **n_calls** because the number of subqueries is not counted in the value of **n_calls**.
n_hard_parse	bigint	Number of hard parsing times. The value of **n_soft_parse** plus the value of **n_hard_parse** may be greater than the value of **n_calls** because the number of subqueries is not counted in the value of **n_calls**.
query_plan	text	Statement execution plan.

Name	Type	Description
n_returned_rows	bigint	Number of rows in the result set returned by the **SELECT** statement.
n_tuples_fetched	bigint	Number of rows randomly scanned.
n_tuples_returned	bigint	Number of rows sequentially scanned.
n_tuples_inserted	bigint	Number of rows inserted.
n_tuples_updated	bigint	Number of rows updated.
n_tuples_deleted	bigint	Number of rows deleted.
n_blocks_fetched	bigint	Number of buffer block access times.
n_blocks_hit	bigint	Number of buffer block hits.
db_time	bigint	Valid DB time, which is accumulated if multiple threads are involved (unit: μs).
cpu_time	bigint	CPU time (unit: μs).
execution_time	bigint	Execution time in the executor (unit: μs).
parse_time	bigint	SQL parsing time (unit: μs).
plan_time	bigint	SQL plan generation time (unit: μs).
rewrite_time	bigint	SQL rewriting time (unit: μs).
pl_execution_time	bigint	Execution time of PL/pgSQL (unit: μs).
pl_compilation_time	bigint	Compilation time of PL/pgSQL (unit: μs).
data_io_time	bigint	I/O time (unit: μs).
net_send_info	text	Network status of messages sent through a physical connection, including the time (unit: μs), number of calls, and throughput (unit: byte). This field can be used to analyze the network overhead of SQL in a distributed system. This field is not supported in a standalone mode. Example: {"time":xxx, "n_calls":xxx, "size":xxx}.

Name	Type	Description
net_recv_info	text	Network status of messages received through a physical connection, including the time (unit: μs), number of calls, and throughput (unit: byte). This field can be used to analyze the network overhead of SQL in a distributed system. This field is not supported in a standalone mode. Example: {"time":xxx, "n_calls":xxx, "size":xxx}.
net_stream_send_info	text	Network status of messages sent through a logical connection, including the time (unit: μs), number of calls, and throughput (unit: byte). This field can be used to analyze the network overhead of SQL in a distributed system. This field is not supported in a standalone mode. Example: {"time":xxx, "n_calls":xxx, "size":xxx}.
net_stream_recv_info	text	Network status of messages received through a logical connection, including the time (unit: μs), number of calls, and throughput (unit: byte). This field can be used to analyze the network overhead of SQL in a distributed system. This field is not supported in a standalone mode. Example: {"time":xxx, "n_calls":xxx, "size":xxx}.
lock_count	bigint	Number of locks.
lock_time	bigint	Time required for locking.
lock_wait_count	bigint	Number of lock waits.
lock_wait_time	bigint	Time required for lock waiting.
lock_max_count	bigint	Maximum number of locks.
lwlock_count	bigint	Number of lightweight locks (reserved).
lwlock_wait_count	bigint	Number of lightweight lock waits.
lwlock_time	bigint	Time required for lightweight locking (reserved).
lwlock_wait_time	bigint	Time required for lightweight locking.

Name	Type	Description
details	bytea	List of statement lock events, which are recorded in time sequence. The number of records is affected by the **track_stmt_details_size** parameter. Events include: ● Start locking. ● Complete locking. ● Start lock waiting. ● Complete lock waiting. ● Start unlocking. ● Complete unlocking. ● Start lightweight lock waiting. ● Complete lightweight lock waiting.
is_slow_sql	boolean	Whether the SQL statement is a slow SQL statement.
trace_id	text	Driver-specific trace ID, which is associated with an application request.
advise	text	Risks which may cause slow SQL statements. (Multiple risks may exist at the same time.) ● Cast Function Cause Index Miss. : Index matching may fail due to implicit conversion. ● Limit too much rows. : The SQL statement execution may slow down due to a large **limit** value. ● Proleakproof of function is false. : The **proleakproof** of the function is set to **false**. In this case, the function does not use statistics when generating a plan due to data leakage risks. As a result, the accuracy of the generated plan is affected and the SQL statement execution may slow down.

14.1.9.15 GS_SLOW_QUERY_INFO (Discarded)

GS_SLOW_QUERY_INFO displays the slow query information that has been dumped on the current node. The data is dumped from the kernel to the system catalog. If **enable_resource_record** is set to **on**, the system imports the query information from the kernel to **GS_WLM_SESSION_QUERY_INFO_ALL** every 3 minutes. This operation occupies storage space and affects performance. You can check **GS_SLOW_QUERY_INFO** to view the slow query information that has been dumped. This view has been discarded in this version.

Table 14-108 GS_SLOW_QUERY_INFO columns

Name	Type	Description
dbname	text	Database name
schemaname	text	Schema name
nodename	text	Node name
username	text	Username
queryid	bigint	Normalization ID
query	text	Query statement
start_time	timestamp with time zone	Execution start time
finish_time	timestamp with time zone	Execution end time
duration	bigint	Execution duration (unit: ms)
query_plan	text	Plan information
n_returned_rows	bigint	Number of rows in the result set returned by the SELECT statement
n_tuples_fetched	bigint	Number of rows randomly scanned
n_tuples_returned	bigint	Number of rows sequentially scanned
n_tuples_inserted	bigint	Number of rows inserted
n_tuples_updated	bigint	Number of rows updated
n_tuples_deleted	bigint	Number of rows deleted
n_blocks_fetched	bigint	Number of cache loading times
n_blocks_hit	bigint	Cache hits
db_time	bigint	Valid DB time, which is accumulated if multiple threads are involved (unit: μs)
cpu_time	bigint	CPU time (unit: μs)
execution_time	bigint	Execution time in the executor (unit: μs)
parse_time	bigint	SQL parsing time (unit: μs)

Name	Type	Description
plan_time	bigint	SQL plan generation time (unit: μs)
rewrite_time	bigint	SQL rewriting time (unit: μs)
pl_execution_time	bigint	Execution time of PL/pgSQL (unit: μs)
pl_compilation_time	bigint	Compilation time of PL/pgSQL (unit: μs)
net_send_time	bigint	Network time (unit: μs)
data_io_time	bigint	I/O time (unit: μs)

14.1.9.16 GS_SLOW_QUERY_HISTORY (Discarded)

GS_SLOW_QUERY_HISTORY displays the slow query information that is not dumped on the current node. For details, see **GS_SLOW_QUERY_INFO**. Only the **system admin** and **monitor admin** users have the permission to query this view. This view is discarded in this version.

14.1.9.17 GLOBAL_SLOW_QUERY_HISTORY (Discarded)

GS_SLOW_QUERY_HISTORY displays the slow query information that is not dumped on all nodes. This view is discarded in this version. For details, see **GS_SLOW_QUERY_INFO**.

14.1.9.18 GLOBAL_SLOW_QUERY_INFO (Discarded)

GS_SLOW_QUERY_HISTORY displays the slow query information that has been dumped on all nodes. This view is discarded in this version. For details, see **GS_SLOW_QUERY_INFO**.

14.1.10 Cache/IO

14.1.10.1 STATIO_USER_TABLES

STATIO_USER_TABLES displays I/O status information about all user relationship tables in the namespace.

Table 14-109 STATIO_USER_TABLES columns

Name	Type	Description
relid	oid	Table OID

Name	Type	Description
schemaname	name	Name of the schema that the table is in
relname	name	Table name
heap_blks_read	bigint	Number of disk blocks read from the table
heap_blks_hit	bigint	Number of cache hits in the table
idx_blks_read	bigint	Number of disk blocks read from indexes in the table
idx_blks_hit	bigint	Number of cache hits in indexes in the table
toast_blks_read	bigint	Number of disk blocks read from the TOAST table (if any) in the table
toast_blks_hit	bigint	Number of buffer hits in the TOAST table (if any) in the table
tidx_blks_read	bigint	Number of disk blocks read from the TOAST table index (if any) in the table
tidx_blks_hit	bigint	Number of buffer-hits in the TOAST table index (if any) in the table

14.1.10.2 SUMMARY_STATIO_USER_TABLES

SUMMARY_STATIO_USER_TABLES displays I/O status information about all user relationship tables in namespaces in the database.

Table 14-110 SUMMARY_STATIO_USER_TABLES columns

Name	Type	Description
schemaname	name	Name of the schema that contains the table
relname	name	Table name
heap_blks_read	numeric	Number of disk blocks read from the table
heap_blks_hit	numeric	Number of cache hits in the table
idx_blks_read	numeric	Number of disk blocks read from all indexes in the table
idx_blks_hit	numeric	Number of cache hits in the table
toast_blks_read	numeric	Number of disk blocks read from the TOAST table (if any) in the table

Name	Type	Description
toast_blks_hit	numeric	Number of buffer hits in the TOAST table (if any) in the table
tidx_blks_read	numeric	Number of disk blocks read from the TOAST table index (if any) in the table
tidx_blks_hit	numeric	Number of buffer-hits in the TOAST table index (if any) in the table

14.1.10.3 GLOBAL_STATIO_USER_TABLES

GLOBAL_STATIO_USER_TABLES displays I/O status information about all user relationship tables in namespaces on each node.

Table 14-111 GLOBAL_STATIO_USER_TABLES columns

Name	Type	Description
node_name	name	Node name
relid	oid	Table OID
schemaname	name	Name of the schema that contains the table
relname	name	Table name
heap_blks_read	bigint	Number of disk blocks read from the table
heap_blks_hit	bigint	Number of cache hits in the table
idx_blks_read	bigint	Number of disk blocks read from all indexes in the table
idx_blks_hit	bigint	Number of cache hits in the table
toast_blks_read	bigint	Number of disk blocks read from the TOAST table (if any) in the table
toast_blks_hit	bigint	Number of buffer hits in the TOAST table (if any) in the table
tidx_blks_read	bigint	Number of disk blocks read from the TOAST table index (if any) in the table
tidx_blks_hit	bigint	Number of buffer-hits in the TOAST table index (if any) in the table

14.1.10.4 STATIO_USER_INDEXES

STATIO_USER_INDEXES displays I/O status information about all user relationship table indexes in namespaces on the current node.

Table 14-112 STATIO_USER_INDEXES columns

Name	Type	Description
relid	oid	OID of the table that the index is created for
indexrelid	oid	OID of the index
schemaname	name	Name of the schema that the index is in
relname	name	Name of the table that the index is created for
indexrelname	name	Index name
idx_blks_read	bigint	Number of disk blocks read from the index
idx_blks_hit	bigint	Number of cache hits in the index

14.1.10.5 SUMMARY_STATIO_USER_INDEXES

SUMMARY_STATIO_USER_INDEXES displays I/O status information about all user relationship table indexes in namespaces in the database.

Table 14-113 SUMMARY_STATIO_USER_INDEXES columns

Name	Type	Description
schemaname	name	Name of the schema that the index is in
relname	name	Name of the table that the index is created for
indexrelname	name	Index name
idx_blks_read	numeric	Number of disk blocks read from the index
idx_blks_hit	numeric	Number of cache hits in the index

14.1.10.6 GLOBAL_STATIO_USER_INDEXES

GLOBAL_STATIO_USER_INDEXES displays I/O status information about all user relationship table indexes in namespaces on each node.

Table 14-114 GLOBAL_STATIO_USER_INDEXES columns

Name	Type	Description
node_name	name	Node name
relid	oid	OID of the table that the index is created for
indexrelid	oid	OID of the index
schemaname	name	Name of the schema that the index is in
relname	name	Name of the table that the index is created for
indexrelname	name	Index name
idx_blks_read	numeric	Number of disk blocks read from the index
idx_blks_hit	numeric	Number of cache hits in the index

14.1.10.7 STATIO_USER_SEQUENCES

STATIO_USER_SEQUENCE displays I/O status information about all user relationship table sequences in namespaces on the current node.

Table 14-115 STATIO_USER_SEQUENCE columns

Name	Type	Description
relid	oid	OID of the sequence
schemaname	name	Name of the schema that the sequence is in
relname	name	Sequence name
blks_read	bigint	Number of disk blocks read from the sequence
blks_hit	bigint	Number of cache hits in the sequence

14.1.10.8 SUMMARY_STATIO_USER_SEQUENCES

SUMMARY_STATIO_USER_SEQUENCES displays I/O status information about all user-defined sequences in namespaces in the database.

Table 14-116 SUMMARY_STATIO_USER_SEQUENCES columns

Name	Type	Description
schemaname	name	Name of the schema that the sequence is in
relname	name	Sequence name
blks_read	numeric	Number of disk blocks read from the sequence
blks_hit	numeric	Number of cache hits in the sequence

14.1.10.9 GLOBAL_STATIO_USER_SEQUENCES

GLOBAL_STATIO_USER_SEQUENCES displays I/O status information about all user-defined sequences in namespaces on each node.

Table 14-117 GLOBAL_STATIO_USER_SEQUENCES columns

Name	Type	Description
node_name	name	Node name
relid	oid	OID of the sequence
schemaname	name	Name of the schema that the sequence is in
relname	name	Sequence name
blks_read	bigint	Number of disk blocks read from the sequence
blks_hit	bigint	Number of cache hits in the sequence

14.1.10.10 STATIO_SYS_TABLES

PG_STATIO_SYS_TABLES displays I/O status information about all system catalogs in the current namespace.

Table 14-118 STATIO_SYS_TABLES columns

Name	Type	Description
relid	oid	Table OID
schemaname	name	Name of the schema that the table is in
relname	name	Table name
heap_blks_read	bigint	Number of disk blocks read from the table

Name	Type	Description
heap_blks_hit	bigint	Number of cache hits in the table
idx_blks_read	bigint	Number of disk blocks read from indexes in the table
idx_blks_hit	bigint	Number of cache hits in indexes in the table
toast_blks_read	bigint	Number of disk blocks read from the TOAST table (if any) in the table
toast_blks_hit	bigint	Number of buffer hits in the TOAST table (if any) in the table
tidx_blks_read	bigint	Number of disk blocks read from the TOAST table index (if any) in the table
tidx_blks_hit	bigint	Number of buffer-hits in the TOAST table index (if any) in the table

14.1.10.11 SUMMARY_STATIO_SYS_TABLES

SUMMARY_STATIO_SYS_TABLES displays I/O status information about all system catalogs in namespaces in the database.

Table 14-119 SUMMARY_STATIO_SYS_TABLES columns

Name	Type	Description
schemaname	name	Name of the schema that contains the table
relname	name	Table name
heap_blks_read	numeric	Number of disk blocks read from the table
heap_blks_hit	numeric	Number of cache hits in the table
idx_blks_read	numeric	Number of disk blocks read from all indexes in the table
idx_blks_hit	numeric	Number of cache hits in the table
toast_blks_read	numeric	Number of disk blocks read from the TOAST table (if any) in the table
toast_blks_hit	numeric	Number of buffer hits in the TOAST table (if any) in the table
tidx_blks_read	numeric	Number of disk blocks read from the TOAST table index (if any) in the table

Name	Type	Description
tidx_blks_hit	numeric	Number of buffer-hits in the TOAST table index (if any) in the table

14.1.10.12 GLOBAL_STATIO_SYS_TABLES

GLOBAL_STATIO_SYS_TABLES displays I/O status information about all system catalogs in namespaces on each node.

Table 14-120 GLOBAL_STATIO_SYS_TABLES columns

Name	Type	Description
node_name	name	Node name
relid	oid	Table OID
schemaname	name	Name of the schema that contains the table
relname	name	Table name
heap_blks_read	bigint	Number of disk blocks read from the table
heap_blks_hit	bigint	Number of cache hits in the table
idx_blks_read	bigint	Number of disk blocks read from all indexes in the table
idx_blks_hit	bigint	Number of cache hits in the table
toast_blks_read	bigint	Number of disk blocks read from the TOAST table (if any) in the table
toast_blks_hit	bigint	Number of buffer hits in the TOAST table (if any) in the table
tidx_blks_read	bigint	Number of disk blocks read from the TOAST table index (if any) in the table
tidx_blks_hit	bigint	Number of buffer-hits in the TOAST table index (if any) in the table

14.1.10.13 STATIO_SYS_INDEXES

STATIO_SYS_INDEXES displays the I/O status information about all system catalog indexes in the current namespace.

Table 14-121 STATIO_SYS_INDEXES columns

Name	Type	Description
relid	oid	OID of the table that the index is created for
indexrelid	oid	OID of the index
schemaname	name	Name of the schema that the index is in
relname	name	Name of the table that the index is created for
indexrelname	name	Index name
idx_blks_read	bigint	Number of disk blocks read from the index
idx_blks_hit	bigint	Number of cache hits in the index

14.1.10.14 SUMMARY_STATIO_SYS_INDEXES

SUMMARY_STATIO_SYS_INDEXES displays I/O status information about all system catalog indexes in namespaces in the database.

Table 14-122 SUMMARY_STATIO_SYS_INDEXES columns

Name	Type	Description
schemaname	name	Name of the schema that the index is in
relname	name	Name of the table that the index is created for
indexrelname	name	Index name
idx_blks_read	numeric	Number of disk blocks read from the index
idx_blks_hit	numeric	Number of cache hits in the index

14.1.10.15 GLOBAL_STATIO_SYS_INDEXES

GLOBAL_STATIO_SYS_INDEXES displays I/O status information about all system catalog indexes in namespaces on each node.

Table 14-123 GLOBAL_STATIO_SYS_INDEXES columns

Name	Type	Description
node_name	name	Node name
relid	oid	OID of the table that the index is created for

Name	Type	Description
indexrelid	oid	OID of the index
schemaname	name	Name of the schema that the index is in
relname	name	Name of the table that the index is created for
indexrelname	name	Index name
idx_blks_read	numeric	Number of disk blocks read from the index
idx_blks_hit	numeric	Number of cache hits in the index

14.1.10.16 STATIO_SYS_SEQUENCES

STATIO_SYS_SEQUENCES shows the I/O status information about all the system sequences in the current namespace.

Table 14-124 STATIO_SYS_SEQUENCES columns

Name	Type	Description
relid	oid	OID of the sequence
schemaname	name	Name of the schema that the sequence is in
relname	name	Sequence name
blks_read	bigint	Number of disk blocks read from the sequence
blks_hit	bigint	Number of cache hits in the sequence

14.1.10.17 SUMMARY_STATIO_SYS_SEQUENCES

SUMMARY_STATIO_SYS_SEQUENCES displays I/O status information about all system sequences in namespaces in the database.

Table 14-125 SUMMARY_STATIO_SYS_SEQUENCES columns

Name	Type	Description
schemaname	name	Name of the schema that the sequence is in
relname	name	Sequence name
blks_read	numeric	Number of disk blocks read from the sequence

Name	Type	Description
blks_hit	numeric	Number of cache hits in the sequence

14.1.10.18 GLOBAL_STATIO_SYS_SEQUENCES

GLOBAL_STATIO_SYS_SEQUENCES displays I/O status information about all system sequences in namespaces on each node.

Table 14-126 GLOBAL_STATIO_SYS_SEQUENCES columns

Name	Type	Description
node_name	name	Node name
relid	oid	OID of the sequence
schemaname	name	Name of the schema that the sequence is in
relname	name	Sequence name
blks_read	bigint	Number of disk blocks read from the sequence
blks_hit	bigint	Number of cache hits in the sequence

14.1.10.19 STATIO_ALL_TABLES

STATIO_ALL_TABLES contains one row for each table (including TOAST tables) in the current database, showing I/O statistics about specific tables.

Table 14-127 STATIO_ALL_TABLES columns

Name	Type	Description
relid	oid	Table OID
schemaname	name	Name of the schema that the table is in
relname	name	Table name
heap_blks_read	bigint	Number of disk blocks read from the table
heap_blks_hit	bigint	Number of cache hits in the table
idx_blks_read	bigint	Number of disk blocks read from all indexes in the table
idx_blks_hit	bigint	Number of cache hits in indexes in the table

Name	Type	Description
toast_blks_read	bigint	Number of disk blocks read from the TOAST table (if any) in the table
toast_blks_hit	bigint	Number of buffer hits in the TOAST table (if any) in the table
tidx_blks_read	bigint	Number of disk blocks read from the TOAST table index (if any) in the table
tidx_blks_hit	bigint	Number of buffer-hits in the TOAST table index (if any) in the table

14.1.10.20 SUMMARY_STATIO_ALL_TABLES

SUMMARY_STATIO_ALL_TABLES contains I/O statistics about each table (including TOAST tables) in databases in the database.

Table 14-128 SUMMARY_STATIO_ALL_TABLES columns

Name	Type	Description
schemaname	name	Name of the schema that contains the table
relname	name	Table name
heap_blks_read	numeric	Number of disk blocks read from the table
heap_blks_hit	numeric	Number of cache hits in the table
idx_blks_read	numeric	Number of disk blocks read from all indexes in the table
idx_blks_hit	numeric	Number of cache hits in the table
toast_blks_read	numeric	Number of disk blocks read from the TOAST table (if any) in the table
toast_blks_hit	numeric	Number of buffer hits in the TOAST table (if any) in the table
tidx_blks_read	numeric	Number of disk blocks read from the TOAST table index (if any) in the table
tidx_blks_hit	numeric	Number of buffer-hits in the TOAST table index (if any) in the table

14.1.10.21 GLOBAL_STATIO_ALL_TABLES

GLOBAL_STATIO_ALL_TABLES contains I/O statistics about each table (including TOAST tables) in databases on each node.

Table 14-129 GLOBAL_STATIO_ALL_TABLES columns

Name	Type	Description
node_name	name	Node name
relid	oid	Table OID
schemaname	name	Name of the schema that contains the table
relname	name	Table name
heap_blks_read	bigint	Number of disk blocks read from the table
heap_blks_hit	bigint	Number of cache hits in the table
idx_blks_read	bigint	Number of disk blocks read from all indexes in the table
idx_blks_hit	bigint	Number of cache hits in the table
toast_blks_read	bigint	Number of disk blocks read from the TOAST table (if any) in the table
toast_blks_hit	bigint	Number of buffer hits in the TOAST table (if any) in the table
tidx_blks_read	bigint	Number of disk blocks read from the TOAST table index (if any) in the table
tidx_blks_hit	bigint	Number of buffer-hits in the TOAST table index (if any) in the table

14.1.10.22 STATIO_ALL_INDEXES

STATIO_ALL_INDEXES contains one row for each index in the current database, showing I/O statistics about specific indexes.

Table 14-130 STATIO_ALL_INDEXES columns

Name	Type	Description
relid	oid	OID of the table that the index is created for
indexrelid	oid	OID of the index
schemaname	name	Name of the schema that the index is in
relname	name	Name of the table that the index is created for
indexrelname	name	Index name

Name	Type	Description
idx_blks_read	bigint	Number of disk blocks read from the index
idx_blks_hit	bigint	Number of cache hits in the index

14.1.10.23 SUMMARY_STATIO_ALL_INDEXES

SUMMARY_STATIO_ALL_INDEXES contains every row of each index in a database in the entire database system, showing I/O statistics about specific indexes.

Table 14-131 SUMMARY_STATIO_ALL_INDEXES columns

Name	Type	Description
schemaname	name	Name of the schema that the index is in
relname	name	Name of the table that the index is created for
indexrelname	name	Index name
idx_blks_read	numeric	Number of disk blocks read from the index
idx_blks_hit	numeric	Number of cache hits in the index

14.1.10.24 GLOBAL_STATIO_ALL_INDEXES

GLOBAL_STATIO_ALL_INDEXES contains one row for each index in databases on each node, showing I/O statistics about specific indexes.

Table 14-132 GLOBAL_STATIO_ALL_INDEXES columns

Name	Type	Description
node_name	name	Node name
relid	oid	OID of the table that the index is created for
indexrelid	oid	OID of the index
schemaname	name	Name of the schema that the index is in
relname	name	Name of the table that the index is created for
indexrelname	name	Index name
idx_blks_read	numeric	Number of disk blocks read from the index

Name	Type	Description
idx_blks_hit	numeric	Number of cache hits in the index

14.1.10.25 STATIO_ALL_SEQUENCES

STATIO_ALL_SEQUENCES contains one row for each sequence in the current database, showing I/O statistics about specific sequences.

Table 14-133 STATIO_ALL_SEQUENCES columns

Name	Type	Description
relid	oid	OID of this sequence
schemaname	name	Name of the schema where the sequence is in
relname	name	Sequence name
blks_read	bigint	Number of disk blocks read from the sequence
blks_hit	bigint	Cache hits in the sequence

14.1.10.26 SUMMARY_STATIO_ALL_SEQUENCES

SUMMARY_STATIO_ALL_SEQUENCES contains one row for each sequence in a database in the entire database system, showing I/O statistics about specific sequences.

Table 14-134 SUMMARY_STATIO_ALL_SEQUENCES columns

Name	Type	Description
schemaname	name	Name of the schema that the sequence is in
relname	name	Sequence name
blks_read	numeric	Number of disk blocks read from the sequence
blks_hit	numeric	Number of cache hits in the sequence

14.1.10.27 GLOBAL_STATIO_ALL_SEQUENCES

GLOBAL_STATIO_ALL_SEQUENCES contains every row of each sequence in databases on each node, showing I/O statistics about specific sequences.

Table 14-135 GLOBAL_STATIO_ALL_SEQUENCES columns

Name	Type	Description
node_name	name	Node name
relid	oid	OID of the sequence
schemaname	name	Name of the schema that the sequence is in
relname	name	Sequence name
blks_read	bigint	Number of disk blocks read from the sequence
blks_hit	bigint	Number of cache hits in the sequence

14.1.10.28 GLOBAL_STAT_DB_CU

GLOBAL_STAT_DB_CU queries CU hits in each database in GaussDB. You can clear it using **pg_stat_reset()**.

Table 14-136 GLOBAL_STAT_DB_CU columns

Name	Type	Description
node_name1	text	Node name
db_name	text	Database name
mem_hit	bigint	Number of memory hits
hdd_sync_read	bigint	Number of synchronous hard disk reads
hdd_asyn_read	bigint	Number of asynchronous hard disk reads

14.1.10.29 GLOBAL_STAT_SESSION_CU

GLOBAL_STAT_SESSION_CU is used to query the CU hit rate of running sessions on each node in the database. The data about a session is cleared when you exit this session or restart the database.

Table 14-137 GLOBAL_STAT_SESSION_CU columns

Name	Type	Description
mem_hit	integer	Number of memory hits
hdd_sync_read	integer	Number of synchronous hard disk reads

Name	Type	Description
hdd_asyn_read	integer	Number of asynchronous hard disk reads

14.1.11 Utility

14.1.11.1 REPLICATION_STAT

REPLICATION_STAT describes information about log synchronization status, such as the locations where the sender sends logs and where the receiver receives logs.

Table 14-138 REPLICATION_STAT columns

Name	Type	Description
pid	bigint	Process ID of the thread
usesysid	oid	User system ID
usename	name	Username
application_name	text	Program name
client_addr	inet	Client address
client_hostname	text	Client name
client_port	integer	Port of the client
backend_start	timestamp with time zone	Start time of the program
state	text	Log replication state: ● Catch-up state ● Consistent streaming state
sender_sent_location	text	Location where the sender sends logs
receiver_write_location	text	Location where the receiver writes logs
receiver_flush_location	text	Location where the receiver flushes logs
receiver_replay_location	text	Location where the receiver replays logs
sync_priority	integer	Priority of synchronous duplication (**0** indicates asynchronization.)

Name	Type	Description
sync_state	text	Synchronization state: ● Asynchronous replication ● Synchronous replication ● Potential synchronization

14.1.11.2 GLOBAL_REPLICATION_STAT

GLOBAL_REPLICATION_STAT displays information about log synchronization status on each node, such as the locations where the sender sends logs and where the receiver receives logs.

Table 14-139 GLOBAL_REPLICATION_STAT columns

Name	Type	Description
node_name	name	Node name
pid	bigint	PID of the thread
usesysid	oid	User system ID
usename	name	Username
application_name	text	Program name
client_addr	inet	Client address
client_hostname	text	Client name
client_port	integer	Port of the client
backend_start	timestamp with time zone	Start time of the program
state	text	Log replication state: ● Catch-up state ● Consistent streaming state
sender_sent_location	text	Location where the transmit sends logs
receiver_write_location	text	Location where the receive end writes logs
receiver_flush_location	text	Location where the receive end flushes logs
receiver_replay_location	text	Location where the receive end replays logs

Name	Type	Description
sync_priority	integer	Priority of synchronous duplication (**0** indicates asynchronization.)
sync_state	text	Synchronization state: ● Asynchronous replication ● Synchronous replication ● Potential synchronization

14.1.11.3 REPLICATION_SLOTS

REPLICATION_SLOTS displays replication slot information.

Table 14-140 REPLICATION_SLOTS columns

Name	Type	Description
slot_name	text	Replication slot name.
plugin	text	Name of the output plug-in corresponding to the logical replication slot.
slot_type	text	Replication slot type. ● **physical**: physical replication slot. ● **logical**: logical replication slot.
datoid	oid	OID of the database where the replication slot resides.
database	name	Name of the database where the replication slot resides.
active	boolean	Determines whether the replication slot is activated. ● **t** (true): yes ● **f** (false): no
xmin	xid	XID of the earliest transaction that the database must reserve for the replication slot.
catalog_xmin	xid	XID of the earliest system catalog-involved transaction that the database must reserve for the logical replication slot.
restart_lsn	text	Physical location of the earliest Xlog required by the replication slot.

Name	Type	Description
dummy_standby	boolean	Determines whether the peer end connected to the replication slot is a secondary node. ● **t** (true): yes ● **f** (false): no

14.1.11.4 GLOBAL_REPLICATION_SLOTS

GLOBAL_REPLICATION_SLOTS displays replication slot information of each node in the database.

Table 14-141 GLOBAL_REPLICATION_SLOTS columns

Name	Type	Description
node_name	name	Node name
slot_name	text	Replication slot name.
plugin	text	Name of the output plug-in corresponding to the logical replication slot.
slot_type	text	Replication slot type. ● **physical**: physical replication slot. ● **logical**: logical replication slot.
datoid	oid	OID of the database where the replication slot resides.
database	name	Name of the database where the replication slot resides.
active	boolean	Determines whether the replication slot is activated. ● **t** (true): yes ● **f** (false): no
x_min	xid	XID of the earliest transaction that the database must reserve for the replication slot.
catalog_xmin	xid	XID of the earliest system catalog-involved transaction that the database must reserve for the logical replication slot.
restart_lsn	text	Physical location of the earliest Xlog required by the replication slot.

Name	Type	Description
dummy_standby	boolean	Determines whether the peer end connected to the replication slot is a secondary node. ● **t** (true): yes ● **f** (false): no

14.1.11.5 BGWRITER_STAT

BGWRITER_STAT displays statistics about the background writer process's activities.

Table 14-142 BGWRITER_STAT columns

Name	Type	Description
checkpoints_timed	bigint	Number of scheduled checkpoints that have been performed
checkpoints_req	bigint	Number of requested checkpoints that have been performed
checkpoint_write_time	double precision	Total time that has been spent in the portion of checkpoint processing where files are written to disk (unit: ms)
checkpoint_sync_time	double precision	Total time that has been spent in the portion of checkpoint processing where files are synchronized to disk (unit: ms)
buffers_checkpoint	bigint	Number of buffers written during checkpoints
buffers_clean	bigint	Number of buffers written by the background writer
maxwritten_clean	bigint	Number of times the background writer stopped a cleaning scan because it had written too many buffers
buffers_backend	bigint	Number of buffers written directly by a backend
buffers_backend_fsync	bigint	Number of times a backend had to execute its own fsync call (normally the background writer handles those even when the backend does its own write)
buffers_alloc	bigint	Number of buffers allocated
stats_reset	timestamp with time zone	Time at which these statistics were last reset

14.1.11.6 GLOBAL_BGWRITER_STAT

GLOBAL_BGWRITER_STAT displays statistics about the background writer process's activities on each node.

Table 14-143 GLOBAL_BGWRITER_STAT columns

Name	Type	Description
node_name	name	Node name
checkpoints_timed	bigint	Number of scheduled checkpoints that have been performed
checkpoints_req	bigint	Number of requested checkpoints that have been performed
checkpoint_write_time	double precision	Total time that has been spent in the portion of checkpoint processing where files are written to disk (unit: ms)
checkpoint_sync_time	double precision	Total time that has been spent in the portion of checkpoint processing where files are synchronized to disk (unit: ms)
buffers_checkpoint	bigint	Number of buffers written during checkpoints
buffers_clean	bigint	Number of buffers written by the background writer
maxwritten_clean	bigint	Number of times the background writer stopped a cleaning scan because it had written too many buffers
buffers_backend	bigint	Number of buffers written directly by a backend
buffers_backend_fsync	bigint	Number of times the backend had to execute its own fsync call (normally the background writer handles those even when the backend does its own write)
buffers_alloc	bigint	Number of buffers allocated
stats_reset	timestamp with time zone	Time at which these statistics were last reset

14.1.11.7 GLOBAL_CKPT_STATUS

GLOBAL_CKPT_STATUS displays the checkpoint information and log flushing information about all instances in the database.

Table 14-144 GLOBAL_CKPT_STATUS columns

Name	Type	Description
node_name	text	Node name
ckpt_redo_point	test	Checkpoint of the current instance
ckpt_clog_flush_num	bigint	Number of Clog flushing pages from the startup time to the current time
ckpt_csnlog_flush_num	bigint	Number of CSN log flushing pages from the startup time to the current time
ckpt_multixact_flush_num	bigint	Number of MultiXact flushing pages from the startup time to the current time
ckpt_predicate_flush_num	bigint	Number of predicate flushing pages from the startup time to the current time
ckpt_twophase_flush_num	bigint	Number of two-phase flushing pages from the startup time to the current time

14.1.11.8 GLOBAL_DOUBLE_WRITE_STATUS

GLOBAL_DOUBLE_WRITE_STATUS displays doublewrite file status of all instances in the database. It consists of the **local_double_write_stat** view of each node. Columns in the view on each node are the same.

Table 14-145 GLOBAL_DOUBLE_WRITE_STATUS columns

Name	Type	Description
node_name	text	Node name
curr_dwn	bigint	Sequence number of the doublewrite file
curr_start_page	bigint	Start page for restoring the doublewrite file
file_trunc_num	bigint	Number of times that the doublewrite file is reused
file_reset_num	bigint	Number of reset times after the doublewrite file is full
total_writes	bigint	Total number of I/Os of the doublewrite file
low_threshold_writes	bigint	Number of I/Os for writing doublewrite files with low efficiency (the number of I/O flushing pages at a time is less than 16)

Name	Type	Description
high_threshold_writes	bigint	Number of I/Os for writing doublewrite files with high efficiency (the number of I/O flushing pages at a time is more than 421)
total_pages	bigint	Total number of pages that are flushed to the doublewrite file area
low_threshold_pages	bigint	Number of pages that are flushed with low efficiency
high_threshold_pages	bigint	Number of pages that are flushed with high efficiency
file_id	bigint	ID of the current doublewrite file

14.1.11.9 GLOBAL_PAGEWRITER_STATUS

GLOBAL_PAGEWRITER_STATUS displays the page flushing information and checkpoint information about all instances in the database.

Table 14-146 GLOBAL_PAGEWRITER_STATUS columns

Name	Type	Description
node_name	text	Node name
pgwr_actual_flush_total_num	bigint	Total number of dirty pages flushed from the startup time to the current time
pgwr_last_flush_num	integer	Number of dirty pages flushed in the previous batch
remain_dirty_page_num	bigint	Estimated number of dirty pages that are not flushed
queue_head_page_rec_lsn	text	**recovery_lsn** of the first dirty page in the dirty page queue of the current instance
queue_rec_lsn	text	**recovery_lsn** of the dirty page queue of the current instance
current_xlog_insert_lsn	text	Write position of Xlogs in the current instance
ckpt_redo_point	text	Checkpoint of the current instance

14.1.11.10 GLOBAL_RECORD_RESET_TIME

GLOBAL_RECORD_RESET_TIME displays statistics about reset time in the database. Restart, primary/standby switchover, and database deletion will cause the time to be reset.

Table 14-147 GLOBAL_RECORD_RESET_TIME columns

Name	Type	Description
node_name	text	Node name
reset_time	timestamp with time zone	Time to be reset

14.1.11.11 GLOBAL_REDO_STATUS

GLOBAL_REDO_STATUS displays the log replay status of all instances in the database.

Table 14-148 GLOBAL_REDO_STATUS columns

Name	Type	Description
node_name	text	Node name
redo_start_ptr	bigint	Start point for replaying the instance logs
redo_start_time	bigint	Start time (UTC) when the instance logs are replayed
redo_done_time	bigint	End time (UTC) when the instance logs are replayed
curr_time	bigint	Current time (UTC) of the instance
min_recovery_point	bigint	Position of the minimum consistency point for the instance logs
read_ptr	bigint	Position for reading the instance logs
last_replayed_read_ptr	bigint	Position for replaying the instance logs
recovery_done_ptr	bigint	Replay position after the instance is started
read_xlog_io_counter	bigint	Number of I/Os when the instance reads and replays logs
read_xlog_io_total_dur	bigint	Total I/O latency when the instance reads and replays logs
read_data_io_counter	bigint	Number of data page I/O reads during replay in the instance
read_data_io_total_dur	bigint	Total I/O latency of data page reads during replay in the instance
write_data_io_counter	bigint	Number of data page I/O writes during replay in the instance

Name	Type	Description
write_data_io_total_dur	bigint	Total I/O latency of data page writes during replay in the instance
process_pending_counter	bigint	Number of synchronization times of log distribution threads during replay in the instance
process_pending_total_dur	bigint	Total synchronization latency of log distribution threads during replay in the instance
apply_counter	bigint	Number of synchronization times of replay threads during replay in the instance
apply_total_dur	bigint	Total synchronization latency of replay threads during replay in the instance
speed	bigint	Log replay rate of the current instance
local_max_ptr	bigint	Maximum number of replay logs received by the local host after the instance is started
primary_flush_ptr	bigint	Log point where the host flushes logs to a disk
worker_info	text	Replay thread information of the instance. If concurrent replay is not enabled, the value is **NULL**.

14.1.11.12 GLOBAL_RECOVERY_STATUS

GLOBAL_RECOVERY_STATUS displays log flow control information about the primary and standby nodes.

Table 14-149 GLOBAL_RECOVERY_STATUS columns

Name	Type	Description
node_name	text	Node name (including the primary and standby nodes)
standby_node_name	text	Name of the standby node
source_ip	text	IP address of the primary node
source_port	integer	Port number of the primary node
dest_ip	text	IP address of the standby node
dest_port	integer	Port number of the standby node

Name	Type	Description
current_rto	bigint	Current log flow control time of the standby node (unit: s)
target_rto	bigint	Expected flow control time of the standby node specified by the corresponding GUC parameter (unit: s)
current_sleep_time	bigint	Sleep time required to achieve the expected flow control time (unit: μs)

14.1.11.13 CLASS_VITAL_INFO

CLASS_VITAL_INFO is used to check whether the OIDs of the same table or index are consistent for WDR snapshots.

Table 14-150 CLASS_VITAL_INFO columns

Name	Type	Description
relid	oid	Table OID
schemaname	name	Schema name
relname	name	Table name
relkind	"char"	Object type. Its value can be: ● **r**: ordinary table ● **t**: TOAST table ● **i**: index

14.1.11.14 USER_LOGIN

USER_LOGIN records the number of user logins and logouts.

Table 14-151 USER_LOGIN columns

Name	Type	Description
node_name	text	Database process name
user_name	text	Username
user_id	integer	User OID (Its value is the same as that of **oid** in **pg_authid**.)
login_counter	bigint	Number of logins
logout_counter	bigint	Number of logouts

14.1.11.15 SUMMARY_USER_LOGIN

SUMMARY_USER_LOGIN records information about user logins and logouts on the primary database node.

Table 14-152 SUMMARY_USER_LOGIN columns

Name	Type	Description
node_name	text	Database process name
user_name	text	Username
user_id	integer	User OID (Its value is the same as that of **oid** in **pg_authid**.)
login_counter	bigint	Number of logins
logout_counter	bigint	Number of logouts

14.1.11.16 GLOBAL_SINGLE_FLUSH_DW_STATUS

GLOBAL_SINGLE_FLUSH_DW_STATUS displays information about doublewrite files eliminated on a single page of all instances in the database. In the displayed information, the information before the slash (/) indicates the page flushing status of the first version, and the information after the slash (/) indicates the page flushing status of the second version.

Table 14-153 GLOBAL_SINGLE_FLUSH_DW_STATUS columns

Name	Type	Description
node_name	text	Instance name
curr_dwn	text	Sequence number of the doublewrite file
curr_start_page	text	Start position of the current doublewrite file
total_writes	text	Total number of data write pages in the current doublewrite file
file_trunc_num	text	Number of times that the doublewrite file is reused
file_reset_num	text	Number of reset times after the doublewrite file is full

14.1.11.17 GLOBAL_CANDIDATE_STATUS

GLOBAL_CANDIDATE_STATUS displays the number of candidate buffers and buffer eviction information of all instances in the database.

Table 14-154 GLOBAL_GET_BGWRITER_STATUS columns

Name	Type	Description
node_name	text	Node name
candidate_slots	integer	Number of pages in the candidate buffer chain of the current normal buffer pool
get_buf_from_list	bigint	Number of times that pages are obtained from the candidate buffer chain during buffer eviction in the current normal buffer pool
get_buf_clock_sweep	bigint	Number of times that pages are obtained from the original eviction solution during buffer eviction in the current normal buffer pool
seg_candidate_slots	integer	Number of pages in the candidate buffer chain of the current segment buffer pool
seg_get_buf_from_list	bigint	Number of times that pages are obtained from the candidate buffer chain during buffer eviction in the current segment buffer pool
seg_get_buf_clock_sweep	bigint	Number of times that pages are obtained from the original eviction solution during buffer eviction in the current segment buffer pool

14.1.12 Lock

14.1.12.1 LOCKS

LOCKS displays information about locks held by each open transaction.

Table 14-155 LOCKS columns

Name	Type	Description
locktype	text	Type of the locked object: **relation**, **extend**, **page**, **tuple**, **transactionid**, **virtualxid**, **object**, **userlock**, or **advisory**
database	oid	OID of the database in which the locked object exists. ● The OID is **0** if the object is a shared object. ● The OID is **NULL** if the object is a transaction ID.

Name	Type	Description
relation	oid	OID of the relationship targeted by the lock. The value is **NULL** if the object is not a relationship or part of a relationship.
page	integer	Page number targeted by the lock within the relationship. The value is **NULL** if the object is not a relationship page or row page.
tuple	smallint	Row number targeted by the lock within the page. The value is **NULL** if the object is not a row.
bucket	integer	Hash bucket ID.
virtualxid	text	Virtual ID of the transaction targeted by the lock. The value is **NULL** if the object is not a virtual transaction ID.
transactionid	xid	ID of the transaction targeted by the lock. The value is **NULL** if the object is not a transaction ID.
classid	oid	OID of the system catalog that contains the object. The value is **NULL** if the object is not a general database object.
objid	oid	OID of the locked object within its system catalog. The value is **NULL** if the object is not a general database object.
objsubid	smallint	Column number for a column in the table (**0** if the object is of other object type and **NULL** if the object is not a general database object)
virtualtransaction	text	Virtual ID of the transaction holding or awaiting this lock
pid	bigint	Logical ID of the server thread holding or awaiting this lock. The value is **NULL** if the lock is held by a prepared transaction.
sessionid	bigint	ID of the session holding or awaiting this lock The value is **NULL** if the lock is held by a prepared transaction.
mode	text	Lock mode held or desired by this thread
granted	Boolean	● The value is **TRUE** if the lock is a held lock. ● The value is **FALSE** if the lock is an awaited lock.
fastpath	Boolean	The value is **TRUE** if the lock is obtained through **fast-path**, and is **FALSE** if the lock is obtained through the main lock table.

Name	Type	Description
locktag	text	Lock information that the session waits for. It can be parsed using the **locktag_decode()** function.
global_sessionid	text	Global session ID

14.1.12.2 GLOBAL_LOCKS

GLOBAL_LOCKS displays information about locks held by open transactions on each node.

Table 14-156 GLOBAL_LOCKS columns

Name	Type	Description
node_name	name	Node name
locktype	text	Type of the locked object: **relation**, **extend**, **page**, **tuple**, **transactionid**, **virtualxid**, **object**, **userlock**, or **advisory**
database	oid	OID of the database in which the locked object exists. ● The OID is **0** if the object is a shared object. ● The OID is **NULL** if the object is a transaction ID.
relation	oid	OID of the relationship targeted by the lock. The value is **NULL** if the object is not a relationship or part of a relationship.
page	integer	Page number targeted by the lock within the relationship. The value is **NULL** if the object is not a relationship page or row page.
tuple	smallint	Row number targeted by the lock within the page. The value is **NULL** if the object is not a row.
bucket	integer	Hash bucket ID
virtualxid	text	Virtual ID of the transaction targeted by the lock. The value is **NULL** if the object is not a virtual transaction ID.
transactionid	xid	ID of the transaction targeted by the lock. The value is **NULL** if the object is not a transaction ID.

Name	Type	Description
classid	oid	OID of the system catalog that contains the object. The value is **NULL** if the object is not a general database object.
objid	oid	OID of the locked object within its system catalog. The value is **NULL** if the object is not a general database object.
objsubid	smallint	Column number for a column in the table (**0** if the target is of other object type and **NULL** if the object is not a general database object)
virtualtransaction	text	Virtual ID of the transaction holding or awaiting this lock
pid	bigint	Logical ID of the server thread holding or awaiting this lock The value is **NULL** if the lock is held by a prepared transaction.
mode	text	Lock mode held or desired by this thread
granted	Boolean	● The value is **TRUE** if the lock is a held lock. ● The value is **FALSE** if the lock is an awaited lock.
fastpath	boolean	The value is **TRUE** if the lock is obtained through **fast-path**, and is **FALSE** if the lock is obtained through the main lock table.

14.1.13 Wait Events

14.1.13.1 WAIT_EVENTS

WAIT_EVENTS displays statistics about wait events on the current node. For details about events, see **Table 13-297**, **Table 13-298**, **Table 13-299** and **Table 13-300**. For details about the impact of each transaction lock on services, see **LOCK**.

Table 14-157 WAIT_EVENTS columns

Name	Type	Description
nodename	text	Database process name
type	text	Event type
event	text	Event name
wait	bigint	Number of waiting times

Name	Type	Description
failed_wait	bigint	Number of waiting failures
total_wait_time	bigint	Total waiting time (unit: μs)
avg_wait_time	bigint	Average waiting time (unit: μs)
max_wait_time	bigint	Maximum waiting time (unit: μs)
min_wait_time	bigint	Minimum waiting time (unit: μs)
last_updated	timestamp with time zone	Last time when the event was updated

14.1.13.2 GLOBAL_WAIT_EVENTS

GLOBAL_WAIT_EVENTS displays statistics about wait events on each node.

Table 14-158 GLOBAL_WAIT_EVENTS columns

Name	Type	Description
nodename	text	Database process name
type	text	Event type
event	text	Event name
wait	bigint	Number of waiting times
failed_wait	bigint	Number of waiting failures
total_wait_time	bigint	Total waiting time (unit: μs)
avg_wait_time	bigint	Average waiting time (unit: μs)
max_wait_time	bigint	Maximum waiting time (unit: μs)
min_wait_time	bigint	Minimum waiting time (unit: μs)
last_updated	timestamp with time zone	Last time when the event was updated

14.1.14 Configuration

14.1.14.1 CONFIG_SETTINGS

CONFIG_SETTINGS displays information about parameters of the running database.

Table 14-159 CONFIG_SETTINGS columns

Name	Type	Description
name	text	Parameter name
setting	text	Current parameter value
unit	text	Implicit unit of the parameter
category	text	Logical group of the parameter
short_desc	text	Brief description of the parameter
extra_desc	text	Detailed description of the parameter
context	text	Context required to set the parameter value, including **internal**, **postmaster**, **sighup**, **backend**, **superuser**, and **user**
vartype	text	Parameter type, including **bool**, **enum**, **integer**, **real**, or **string**
source	text	Method of assigning the parameter value
min_val	text	Maximum value of the parameter. If the parameter type is not numeric, the value of this column is **null**.
max_val	text	Minimum value of the parameter. If the parameter type is not numeric, the value of this column is **null**.
enumvals	text[]	Valid values of an enum-type parameter. If the parameter type is not enum, the value of this column is **null**.
boot_val	text	Default parameter value used upon the database startup
reset_val	text	Default parameter value used upon the database reset
sourcefile	text	Configuration file used to set parameter values. If parameter values are not configured using the configuration file, the value of this column is **null**.
sourceline	integer	Row number in the configuration file for setting parameter values. If parameter values are not configured using the configuration file, the value of this column is **null**.

14.1.14.2 GLOBAL_CONFIG_SETTINGS

GLOBAL_CONFIG_SETTINGS displays information about parameters of running databases on each node.

Table 14-160 GLOBAL_CONFIG_SETTINGS columns

Name	Type	Description
node_name	text	Node name
name	text	Parameter name
setting	text	Current parameter value
unit	text	Implicit unit of the parameter
category	text	Logical group of the parameter
short_desc	text	Brief description of the parameter
extra_desc	text	Detailed description of the parameter
context	text	Context required to set the parameter value, including **internal**, **postmaster**, **sighup**, **backend**, **superuser**, and **user**
vartype	text	Parameter type, including **bool**, **enum**, **integer**, **real**, or **string**
source	text	Method of assigning the parameter value
min_val	text	Minimum value of the parameter. If the parameter type is not numeric, the value of this column is **null**.
max_val	text	Maximum value of the parameter. If the parameter type is not numeric, the value of this column is **null**.
enumvals	text[]	Valid values of an enum-type parameter. If the parameter type is not enum, the value of this column is **null**.
boot_val	text	Default parameter value used upon the database startup
reset_val	text	Default parameter value used upon the database reset
sourcefile	text	Configuration file used to set parameter values. If parameter values are not configured using the configuration file, the value of this column is **null**.
sourceline	integer	Row number in the configuration file for setting parameter values. If parameter values are not configured using the configuration file, the value of this column is **null**.

14.1.15 Operator

14.1.15.1 OPERATOR_HISTORY_TABLE

OPERATOR_HISTORY_TABLE displays records about operators of completed jobs. Data is dumped from the kernel to this system catalog.

Table 14-161 OPERATOR_HISTORY_TABLE columns

Name	Type	Description
queryid	bigint	Internal query ID used for statement execution
pid	bigint	Thread ID of the backend
plan_node_id	integer	Plan node ID of the execution plan
plan_node_name	text	Name of the operator corresponding to the plan node ID
start_time	timestamp with time zone	Time when the operator starts to process the first data record
duration	bigint	Total execution time of the operator, in ms
query_dop	integer	DOP of the operator
estimated_rows	bigint	Number of rows estimated by the optimizer
tuple_processed	bigint	Number of elements returned by the operator
min_peak_memory	integer	Minimum peak memory used by the operator on database nodes (unit: MB)
max_peak_memory	integer	Maximum peak memory used by the operator on database nodes (unit: MB)
average_peak_memory	integer	Average peak memory used by the operator on database nodes (unit: MB)
memory_skew_percent	integer	Memory usage skew of the operator among database nodes
min_spill_size	integer	Minimum spilled data among database nodes when a spill occurs (unit: MB) (default value: **0**)
max_spill_size	integer	Maximum spilled data among database nodes when a spill occurs (unit: MB) (default value: **0**)

Name	Type	Description
average_spill_size	integer	Average spilled data among database nodes when a spill occurs (unit: MB) (default value: **0**)
spill_skew_percent	integer	Database node spill skew when a spill occurs
min_cpu_time	bigint	Minimum execution time of the operator on database nodes (unit: ms)
max_cpu_time	bigint	Maximum execution time of the operator on database nodes (unit: ms)
total_cpu_time	bigint	Total execution time of the operator on database nodes (unit: ms)
cpu_skew_percent	integer	Execution time skew among database nodes
warning	text	Warning. The following warnings are displayed: ● Sort/SetOp/HashAgg/HashJoin spill ● Spill file size large than 256MB ● Broadcast size large than 100MB ● Early spill ● Spill times is greater than 3 ● Spill on memory adaptive ● Hash table conflict

14.1.15.2 OPERATOR_HISTORY

OPERATOR_HISTORY displays records of operators in jobs that have been executed by the current user on the current primary database node. Columns in this view are the same as those in **Table 13-32**.

14.1.15.3 OPERATOR_RUNTIME

OPERATOR_RUNTIME displays information about operators of the jobs that are being executed by the current user.

Table 14-162 OPERATOR_RUNTIME columns

Name	Type	Description
queryid	bigint	Internal query ID used for statement execution
pid	bigint	Thread ID of the backend
plan_node_id	integer	Plan node ID of the execution plan of a query

Name	Type	Description
plan_node_na me	text	Name of the operator corresponding to the plan node ID
start_time	timestamp with time zone	Time when the operator starts to process the first data record
duration	bigint	Total execution time of the operator, in ms
status	text	Execution status of the current operator, which can be **finished** or **running**.
query_dop	integer	DOP of the operator
estimated_rows	bigint	Number of rows estimated by the optimizer
tuple_processe d	bigint	Number of elements returned by the operator
min_peak_mem ory	integer	Minimum peak memory used by the operator on database nodes (unit: MB)
max_peak_me mory	integer	Maximum peak memory used by the operator on database nodes (unit: MB)
average_peak_ memory	integer	Average peak memory used by the operator on database nodes (unit: MB)
memory_skew_ percent	integer	Memory usage skew of the operator among database nodes
min_spill_size	integer	Minimum spilled data among database nodes when a spill occurs (unit: MB) (default value: **0**)
max_spill_size	integer	Maximum spilled data among database nodes when a spill occurs (unit: MB) (default value: **0**)
average_spill_si ze	integer	Average spilled data among database nodes when a spill occurs (unit: MB) (default value: **0**)
spill_skew_perc ent	integer	Database node spill skew when a spill occurs
min_cpu_time	bigint	Minimum execution time of the operator on database nodes (unit: ms)
max_cpu_time	bigint	Maximum execution time of the operator on database nodes (unit: ms)
total_cpu_time	bigint	Total execution time of the operator on database nodes (unit: ms)

Name	Type	Description
cpu_skew_percent	integer	Execution time skew among database nodes
warning	text	Warning. The following warnings are displayed: ● Sort/SetOp/HashAgg/HashJoin spill ● Spill file size large than 256MB ● Broadcast size large than 100MB ● Early spill ● Spill times is greater than 3 ● Spill on memory adaptive ● Hash table conflict

14.1.15.4 GLOBAL_OPERATOR_HISTORY

GLOBAL_OPERATOR_HISTORY displays records of operators in jobs that have been executed by the current user on the primary database node.

Table 14-163 GLOBAL_OPERATOR_HISTORY columns

Name	Type	Description
queryid	bigint	Internal query ID used for statement execution
pid	bigint	Thread ID of the backend
plan_node_id	integer	Plan node ID of the execution plan
plan_node_name	text	Name of the operator corresponding to the plan node ID
start_time	timestamp with time zone	Time when the operator starts to process the first data record
duration	bigint	Total execution time of the operator, in ms
query_dop	integer	DOP of the operator
estimated_rows	bigint	Number of rows estimated by the optimizer
tuple_processed	bigint	Number of elements returned by the operator
min_peak_memory	integer	Minimum peak memory used by the operator on database nodes (unit: MB)
max_peak_memory	integer	Maximum peak memory used by the operator on database nodes (unit: MB)

Name	Type	Description
average_peak_memory	integer	Average peak memory used by the operator on database nodes (unit: MB)
memory_skew_percent	integer	Memory usage skew of the operator among database nodes
min_spill_size	integer	Minimum spilled data among database nodes when a spill occurs (unit: MB) (default value: **0**)
max_spill_size	integer	Maximum spilled data among database nodes when a spill occurs (unit: MB) (default value: **0**)
average_spill_size	integer	Average spilled data among database nodes when a spill occurs (unit: MB) (default value: **0**)
spill_skew_percent	integer	Database node spill skew when a spill occurs
min_cpu_time	bigint	Minimum execution time of the operator on database nodes (unit: ms)
max_cpu_time	bigint	Maximum execution time of the operator on database nodes (unit: ms)
total_cpu_time	bigint	Total execution time of the operator on database nodes (unit: ms)
cpu_skew_percent	integer	Execution time skew among database nodes
warning	text	Warning. The following warnings are displayed: 1. Sort/SetOp/HashAgg/HashJoin spill 2. Spill file size large than 256MB 3. Broadcast size large than 100MB 4. Early spill 5. Spill times is greater than 3 6. Spill on memory adaptive 7. Hash table conflict

14.1.15.5 GLOBAL_OPERATOR_HISTORY_TABLE

GLOBAL_OPERATOR_HISTORY_TABLE displays the records about operators of completed jobs on the primary database node. Data is dumped from the kernel to the system catalog **GS_WLM_OPERATOR_INFO**.
GLOBAL_OPERATOR_HISTORY_TABLE is a collection view for querying the system

catalog **GS_WLM_OPERATOR_INFO** on the primary database node. Columns in this view are the same as those in **Table 14-163**.

14.1.15.6 GLOBAL_OPERATOR_RUNTIME

GLOBAL_OPERATOR_RUNTIME displays information about operators of the jobs that are being executed by the current user on the primary database node.

Table 14-164 GLOBAL_OPERATOR_RUNTIME columns

Name	Type	Description
queryid	bigint	Internal query ID used for statement execution
pid	bigint	Thread ID of the backend
plan_node_id	integer	Plan node ID of the execution plan
plan_node_name	text	Name of the operator corresponding to the plan node ID
start_time	timestamp with time zone	Time when the operator starts to process the first data record
duration	bigint	Total execution time of the operator, in ms
status	text	Execution status of the current operator, which can be **finished** or **running**.
query_dop	integer	DOP of the operator
estimated_rows	bigint	Number of rows estimated by the optimizer
tuple_processed	bigint	Number of elements returned by the operator
min_peak_memory	integer	Minimum peak memory used by the operator on database nodes (unit: MB)
max_peak_memory	integer	Maximum peak memory used by the operator on database nodes (unit: MB)
average_peak_memory	integer	Average peak memory used by the operator on database nodes (unit: MB)
memory_skew_percent	integer	Memory usage skew of the operator among database nodes
min_spill_size	integer	Minimum spilled data among database nodes when a spill occurs (unit: MB) (default value: **0**)
max_spill_size	integer	Maximum spilled data among database nodes when a spill occurs (unit: MB) (default value: **0**)

Name	Type	Description
average_spill_size	integer	Average spilled data among database nodes when a spill occurs (unit: MB) (default value: **0**)
spill_skew_percent	integer	Database node spill skew when a spill occurs
min_cpu_time	bigint	Minimum execution time of the operator on database nodes (unit: ms)
max_cpu_time	bigint	Maximum execution time of the operator on database nodes (unit: ms)
total_cpu_time	bigint	Total execution time of the operator on database nodes (unit: ms)
cpu_skew_percent	integer	Execution time skew among database nodes
warning	text	Warning. The following warnings are displayed: ● Sort/SetOp/HashAgg/HashJoin spill ● Spill file size large than 256MB ● Broadcast size large than 100MB ● Early spill ● Spill times is greater than 3 ● Spill on memory adaptive ● Hash table conflict

14.1.16 Workload Manager

14.1.16.1 WLM_USER_RESOURCE_CONFIG

WLM_USER_RESOURCE_CONFIG displays the resource configuration information of a user.

Table 14-165 WLM_USER_RESOURCE_CONFIG columns

Name	Type	Description
userid	oid	OID of the user
username	name	Username
sysadmin	boolean	Whether the user has the **sysadmin** permission
rpoid	oid	OID of the resource pool

Name	Type	Description
respool	name	Name of a resource pool
parentid	oid	OID of the parent user
totalspace	bigint	Size of the occupied space
spacelimit	bigint	Upper limit of the space size
childcount	integer	Number of child users
childlist	text	Child user list

14.1.16.2 WLM_USER_RESOURCE_RUNTIME

WLM_USER_RESOURCE_RUNTIME displays resource usage of all users. Only administrators can query this view. This view is valid only when the GUC parameter **use_workload_manager** is set to **on**.

Table 14-166 WLM_USER_RESOURCE_RUNTIME columns

Name	Type	Description
username	name	Username
used_memory	integer	Used memory (unit: MB)
total_memory	integer	Available memory (unit: MB) The value **0** indicates that the available memory is not limited and depends on the maximum memory available in the database.
used_cpu	integer	Number of CPU cores in use
total_cpu	integer	Total number of CPU cores of the Cgroup associated with the user on the node
used_space	bigint	Used storage space (unit: KB)
total_space	bigint	Available storage space (unit: KB) The value **-1** indicates that the maximum storage space is not limited.
used_temp_space	bigint	Used temporary space (reserved column; unit: KB)
total_temp_space	bigint	Available temporary storage space (reserved column; unit: KB) The value **-1** indicates that the maximum temporary storage space is not limited.
used_spill_space	bigint	Used space for storing spilled data (reserved column; unit: KB)

Name	Type	Description
total_spill_space	bigint	Available storage space for spilled data (reserved column; unit: KB) The value **-1** indicates that the maximum space for spilled data is not limited.

14.1.17 Global Plancache

Global plan cache (GPC) views are valid only when **enable_global_plancache** and the thread pool are enabled. The current feature is a lab feature. Contact Huawei technical support before using it.

14.1.17.1 GLOBAL_PLANCACHE_STATUS

GLOBAL_PLANCACHE_STATUS displays the GPC status information. The current feature is a lab feature. Contact Huawei technical support before using it.

Table 14-167 GLOBAL_PLANCACHE_STATUS columns

Name	Type	Description
nodename	text	Name of the node that the plan cache belongs to
query	text	Text of query statements
refcount	integer	Number of times that the plan cache is referenced
valid	bool	Whether the plan cache is valid
databaseid	oid	ID of the database that the plan cache belongs to
schema_name	text	Schema that the plan cache belongs to
params_num	integer	Number of parameters
func_id	oid	OID of the stored procedure where the plan cache is located. If the plancache does not belong to the stored procedure, the value is **0**.

14.1.17.2 GLOBAL_PLANCACHE_CLEAN

GLOBAL_PLANCACHE_CLEAN clears the global plan cache that is not used on all nodes. The return value is of the Boolean type.

14.1.18 RTO & RPO

14.1.18.1 global_rto_status

Displays log flow control information about the primary and standby nodes (except the current node and standby DNs).

Table 14-168 global_rto_status columns

Parameter	Type	Description
node_name	text	Node name (including the primary and standby nodes)
rto_info	text	Flow control information, including the current log flow control time (unit: second) of the standby node, the expected flow control time (unit: second) specified by the GUC parameter, and the primary node sleep time (unit: μs) required to reach the expectation

14.1.18.2 global_streaming_hadr_rto_and_rpo_stat

global_streaming_hadr_rto_and_rpo_stat displays the log flow control information about the primary and standby database instances in the streaming DR scenario. (This schema can be used only on the primary DN of the primary database instance. Statistics cannot be obtained from the standby DN or the standby database instance.)

Table 14-169 Parameters

Parameter	Type	Description
hadr_sender_node_name	text	Node name, including the primary database instance and the first standby node of the standby database instance.
hadr_receiver_node_name	text	Name of the first standby node of the standby database instance.
current_rto	int	Flow control information, that is, log RTO time of the current primary and standby database instances (unit: second).
target_rto	int	Flow control information, that is, the RTO time between the target primary and standby database instances (unit: second).
current_rpo	int	Flow control information, that is, log RPO time of the current primary and standby database instances (unit: second).
target_rpo	int	Flow control information, that is, the RPO time between the target primary and standby database instances (unit: second).

Parameter	Type	Description
rto_sleep_time	int	RTO flow control information, that is, the expected sleep time (unit: μs) required by walsender on the host to reach the specified RTO.
rpo_sleep_time	int	RPO flow control information, that is, the expected sleep time (unit: μs) required by xlogInsert on the host to reach the specified RPO.

14.2 WDR Snapshot Schema

After the WDR snapshot function is enabled (**enable_wdr_snapshot** is set to **on**), objects are created in schema **snapshot** in user tablespace **pg_default** in database **postgres** to flush WDR snapshot data. By default, the initial user or users with the monadmin permission can access objects in the snapshot schema.

You can set the parameter **wdr_snapshot_retention_days** to automatically manage the snapshot lifecycle.

NOTICE

Do not add, delete, or modify tables in the **snapshot** schema. Manual modification or damage to these tables may cause WDR exceptions or even WDR unavailability.

14.2.1 Original Information of WDR Snapshots

14.2.1.1 SNAPSHOT.SNAPSHOT

SNAPSHOT records the index information, start time, and end time of WDR snapshots stored in the current system. The result can be queried only in the system database. After the WDR snapshot function is enabled (the **enable_wdr_snapshot** parameter is set to **on**), the table is created.

Table 14-170 SNAPSHOT attributes

Name	Type	Description	Example
snapshot_id	bigint	WDR snapshot ID	1
start_ts	timestamp	Start time of a WDR snapshot	2019-12-28 17:11:27.423742+08
end_ts	timestamp	End time of a WDR snapshot	2019-12-28 17:11:43.67726+08

14.2.1.2 SNAPSHOT.TABLES_SNAP_TIMESTAMP

The **TABLES_SNAP_TIMESTAMP** table records the databases, table objects, and start and end time of all WDR snapshots. After the WDR snapshot is started (the **enable_wdr_snapshot** parameter is set to **on**), the table is created.

Table 14-171 TABLES_SNAP_TIMESTAMP attributes

Name	Type	Description	Example
snapshot_id	bigint	WDR snapshot ID	1
db_name	text	Database corresponding to a WDR snapshot	tpcc1000
tablename	text	Table corresponding to a WDR snapshot	snap_xc_statio_all_indexes
start_ts	timestamp	Start time of a WDR snapshot	2019-12-28 17:11:27.425849+08
end_ts	timestamp	End time of a WDR snapshot	2019-12-28 17:11:27.707398+08

14.2.1.3 SNAP_SEQ

SNAP_SEQ is an ascending sequence, which provides IDs for WDR snapshots.

14.2.2 WDR Snapshot Data Table

The naming rule of a WDR snapshot data table is **snap_***{Source data table}*.

WDR snapshot data tables come from all views in **DBE_PERF Schema**.

14.2.3 Performance Report Generated Based on WDR Snapshots

Performance reports are generated based on the summary and statistics of WDR snapshot data tables. By default, the initial user or monitor administrator can generate reports.

Prerequisites

The WDR snapshots have been started (**enable_wdr_snapshot** is set to **on**) and the number of snapshots is greater than or equal to 2.

Procedure

Step 1 Run the following command to create a report file:

```
touch  /home/om/wdrTestNode.html
```

Step 2 Run the following command to connect to the **postgres** database.

```
gsql -d postgres -p [Port number] -r
```

Step 3 Run the following command to query the generated snapshot and obtain **snapshot_id**:

```
select * from snapshot.snapshot;
```

Step 4 (Optional) Run the following command to manually create a snapshot. If only one snapshot exists in the database or you want to view the monitoring data of the database in the current period, manually create a snapshot. This command is only available to users with the sysadmin permission.

```
select create_wdr_snapshot();
```

Step 5 Run the following command to generate a WDR in HTML format on the local PC:

1. Run the following command to set the report format: **\a** indicates that table row and column symbols are not displayed. **\t** indicates that column names are not displayed. **\o** specifies an output file.
```
gsql> \a
gsql> \t
gsql> \o /home/om/wdrTestNode.html
```
2. Run the following command to generate a WDR in HTML format:
```
gsql> select generate_wdr_report(begin_snap_id Oid, end_snap_id Oid, int report_type, int
report_scope, int node_name );
```
Example 1: Generate a database instance-level report.
```
select generate_wdr_report(1, 2, 'all', 'cluster',null);
```
Example 2: Generate a report for a node.
```
select generate_wdr_report(1, 2, 'all', 'node', pgxc_node_str()::cstring);
```

NOTE

Currently, the name of the centralizedopenGauss node is fixed to **dn_6001_6002_6003**. You can also replace it with the actual node name.

Table 14-172 Parameters of the generate_wdr_report function

Parameter	Description	Value Range
begin_snap_id	ID of a snapshot when a query starts, which is specified by **snapshot_id** in the **snapshot.snaoshot** table.	-
end_snap_id	ID of a snapshot when a query ends. By default, the value of **end_snap_id** is greater than that of **begin_snap_id table** (**snapshot_id** in the **snapshot.snaoshot** table).	-

Parameter	Description	Value Range
report_type	Type of the generated report. The value can be **summary**, **detail**, or **all**.	– **summary**: summary data – **detail**: detailed data – **all**: summary data and detailed data
report_scope	Range of the generated report. The value can be **cluster** or **node**.	– **cluster**: database-level information – **node**: node-level information
node_name	When **report_scope** is set to **node**, set this parameter to the name of the corresponding node. (You can run the **select * from pg_node_env;** command to query the node name.) If **report_scope** is set to **cluster**, this parameter can be omitted, left blank, or set to **NULL**.	– **node**: a node name in GaussDB – **cluster**: This value is omitted, left blank or set to **NULL**.

CAUTION

The two snapshots used to generate the report should meet the following conditions:

- No node is restarted between two snapshots.
- No primary/standby switchover is performed between two snapshots.
- Performance indicators cannot be reset between two snapshots.
- No DROP operation is performed on the database between two snapshots.
- If a negative value exists in the generated WDR, it indicates that the indicator cannot reflect the performance of the database.
- The time required for generating a report depends on the amount of performance data in the performance snapshot. Generally, a report can be generated in minutes. If the report cannot be generated within 5 minutes, collect the statistics **ANALYZE | ANALYSE** about the tables in the **snapshot** schema (considering the **snap_global_statio_all_tables** and **snap_global_statio_all_indexes** tables first), and then generate the report again. Alternatively, set **set statement_timeout** to * to terminate report generation.
- When generating a report, ensure that the character set of the client is the same as that of the GaussDB database. (You can run **set client_encoding to *** to set the character set of the client).

3. Disable the output options and format the output.
```
\o \a \t
```

Step 6 **View the WDR** in **/home/om/** as required.

----End

Examples

```
-- Create a report file.
touch  /home/om/wdrTestNode.html

-- Connect to the database.
gsql -d postgres -p [Port number] -r

-- Query the snapshots that have been generated.
openGauss=# select * from snapshot.snapshot;
 snapshot_id |           start_ts            |            end_ts
-------------+-------------------------------+-------------------------------
           1 | 2020-09-07 10:20:36.763244+08 | 2020-09-07 10:20:42.166511+08
           2 | 2020-09-07 10:21:13.416352+08 | 2020-09-07 10:21:19.470911+08
(2 rows)


-- Generate the formatted performance report wdrTestNode.html.
openGauss=# \a \t \o /home/om/wdrTestNode.html
Output format is unaligned.
Showing only tuples.

-- Write data into the performance report wdrTestNode.html.
openGauss=# select generate_wdr_report(1, 2, 'all', 'node', 'dn_6001_6002_6003');

-- Close the performance report wdrTestNode.html.
openGauss=# \o

-- Generate the formatted performance report wdrTestCluster.html.
```

```
openGauss=# \o /home/om/wdrTestCluster.html

-- Write data into the performance report wdrTestCluster.html.
openGauss=# select generate_wdr_report(1, 2, 'all', 'cluster');

-- Close the performance report wdrTestCluster.html.
openGauss=# \o \a \t
Output format is aligned.
Tuples only is off.
```

14.2.4 WDRs

The following table describes the main contents of WDR reports.

Table 14-173 Content of the WDR

Metric	Description
Database Stat	• Database performance statistics: transactions, read and write, row activities, write conflicts, and deadlocks • Database-wide report, which can be viewed only in cluster mode.
Load Profile	• Datebase performance statistics: CPU time, DB time, logical or physical read, I/O performance, login and logout, load strength, and load performance • Database-wide report, which can be viewed only in cluster mode.
Instance Efficiency Percentages	• Database-level or node-level cache hit ratio. • Database/Node-wide report, which can be viewed in cluster or node mode.
Top 10 Events by Total Wait Time	• Most time-consuming event. • Node-wide report, which can be viewed in node mode.
Wait Classes by Total Wait Time	• Category of the wait time that is most time-consuming. • Node-wide report, which can be viewed in node mode.
Host CPU	• CPU usage of the host. • Node-wide report, which can be viewed in node mode.
IO Profile	• I/O usage in the database or node dimension. • Database/Node-wide report, which can be viewed in cluster or node mode.
Memory Statistics	• Kernel memory usage. • Node-wide report, which can be viewed in node mode.
Time Model	• Time distribution information about the statements in a node. • Node-wide report, which can be viewed in node mode.

Metric	Description
SQL Statistics	• SQL statement performance statistics: end-to-end time, row activities, cache hit, CPU consumption, and time consumption. • Database/Node-wide report, which can be viewed in cluster or node mode.
Wait Events	• Statistics on wait events at the node level. • Node-wide report, which can be viewed in node mode.
Cache IO Stats	• I/O statistics on user tables and indexes. • Database/Node-wide report, which can be viewed in cluster or node mode.
Utility status	• Status information about the replication slot and background checkpoint. • Node-wide report, which can be viewed in node mode.
Object stats	• Performance statistics on the index and table dimensions. • Database/Node-wide report, which can be viewed in cluster or node mode.
Configuration settings	• Node configuration. • Node-wide report, which can be viewed in node mode.
SQL Detail	• SQL statement text details. • Database/Node-wide report, which can be viewed in cluster or node mode.

14.2.4.1 Database Stat

The following table describes columns in the Database Stat report.

Table 14-174 Columns in the Database Stat report

Column	Description
DB Name	Database name.
Backends	Number of backends connected to this database.
Xact Commit	Number of transactions in this database that have been committed.
Xact Rollback	Number of transactions in this database that have been rolled back.
Blks Read	Number of disk blocks read in this database.
Blks Hit	Number of times that disk blocks have been found in the cache.

Column	Description
Tuple Returned	Number of rows sequentially scanned.
Tuple Fetched	Number of rows randomly scanned.
Tuple Inserted	Number of rows inserted by queries in this database.
Tuple Updated	Number of rows updated by queries in this database.
Tup Deleted	Number of rows deleted by queries in this database.
Conflicts	Number of queries canceled due to conflicts.
Temp Files	Number of temporary files created by queries in this database.
Temp Bytes	Total amount of data written to temporary files by queries in this database.
Deadlocks	Number of deadlocks detected in this database.
Blk Read Time	Time spent reading data file blocks by backends in this database (unit: ms).
Blk Write Time	Time spent writing into data file blocks by backends in this database (unit: ms).
Stats Reset	Time at which the current statistics were reset.

14.2.4.2 Load Profile

The following table lists metrics in the Load Profile report.

Table 14-175 Metrics in the Load Profile report

Metric	Description
DB Time(us)	Total elapsed time of a job.
CPU Time(us)	Total CPU time used for job running.
Redo size(blocks)	Size of the generated WAL (blocks).
Logical read (blocks)	Number of logical reads (blocks) on a table or an index.
Physical read (blocks)	Number of physical reads (blocks) on a table or an index.
Physical write (blocks)	Number of physical writes (blocks) on a table or an index.

Metric	Description
Read IO requests	Number of times that a table or an index is read.
Write IO requests	Number of times that a table or an index is written.
Read IO (MB)	Table or index read size (MB).
Write IO (MB)	Table or index write size (MB).
Logons	Number of logins.
Executes (SQL)	Number of times the SQL statement is executed.
Rollbacks	Number of rolled-back transactions.
Transactions	Number of transactions.
SQL response time P95(us)	Response time of 95% SQL statements.
SQL response time P80(us)	Response time of 80% SQL statements.

14.2.4.3 Instance Efficiency Percentages

The following table lists metrics in the Instance Efficiency Percentages report.

Table 14-176 Metrics in the Instance Efficiency Percentages report

Metric	Description
Buffer Hit %	Hit ratio of the buffer pool.
Effective CPU %	Ratio of the CPU time to the DB time.
WalWrite NoWait %	Ratio of the number of events that access the WAL buffer to the total number of wait events.
Soft Parse %	Ratio of the number of soft parsing times to the total number of parsing times.
Non-Parse CPU %	Ratio of the non-parse time to the total execution time.

14.2.4.4 Top 10 Events by Total Wait Time

The following table lists columns in the Top 10 Events by Total Wait Time report.

Table 14-177 Columns in the Top 10 Events by Total Wait Time report

Column	Description
Event	Name of a wait event.
Waits	Number of wait times.
Total Wait Time(us)	Total wait time, in microseconds.
Avg Wait Time(us)	Average wait time, in microseconds.
Type	Wait event type.

14.2.4.5 Wait Classes by Total Wait Time

The following table lists columns in the Wait Classes by Total Wait Time report.

Table 14-178 Columns in the Wait Classes by Total Wait Time report

Column	Description
Type	Wait events are classified as follows: ● STATUS ● LWLOCK_EVENT ● LOCK_EVENT ● IO_EVENT
Waits	Number of wait times.
Total Wait Time(us)	Total wait time, in microseconds.
Avg Wait Time(us)	Average wait time, in microseconds.

14.2.4.6 Host CPU

The following table describes columns in the Host CPU report.

Table 14-179 Columns in the Host CPU report

Column	Description
Cpus	Number of processors.
Cores	Number of CPU cores.
Sockets	Number of CPU sockets.

Column	Description
Load Average Begin	Load average value of the start snapshot.
Load Average End	Load average value of the end snapshot.
%User	Percentage of CPU time spent in the user mode.
%System	Percentage of CPU time spent in the kernel mode.
%WIO	Percentage of CPU time spent in wait I/O.
%Idle	Percentage of CPU idle time.

14.2.4.7 IO Profile

The following table lists metrics in the IO Profile report.

Table 14-180 Metrics in the IO Profile report

Metric	Description
Database requests	Number of database I/O times.
Database (MB)	Database I/O data volume.
Database (blocks)	Number of database I/O data blocks.
Redo requests	Number of redo I/O times.
Redo (MB)	Redo I/O data volume.

14.2.4.8 Memory Statistics

The following table lists metrics in the Memory Statistics report.

Table 14-181 Columns in the Memory Statistics report

Metric	Description
shared_used_memory	Size of used shared memory (MB).

Metric	Description
max_shared_memory	Maximum shared memory (MB).
process_used_memory	Memory used by processes (MB).
max_process_memory	Maximum memory that can be allocated to a process (MB).

14.2.4.9 Time Model

The following table describes metrics in the Time Model report.

Table 14-182 Metrics in the Time Model report

Metric	Description
DB_TIME	Total end-to-end wall time consumed by all threads (unit: μs).
EXECUTION_TIME	Total time consumed on the executor (unit: μs).
PL_EXECUTION_TIME	Total time consumed for executing PL/SQL statements (unit: μs).
CPU_TIME	Total CPU time consumed by all threads (unit: μs).
PLAN_TIME	Total time consumed for generating an execution plan (unit: μs).
REWRITE_TIME	Total time consumed on query rewriting (unit: μs).
PL_COMPILATION_TIME	Total time consumed for SQL compilation (unit: μs).
PARSE_TIME	Total time consumed for parsing SQL statements (unit: μs).
NET_SEND_TIME	Total time consumed for sending data over network (unit: μs).
DATA_IO_TIME	Total time consumed for data read and write (unit: μs).

14.2.4.10 SQL Statistics

The following table describes columns in the SQL Statistics report.

Table 14-183 Columns in the SQL Statistics report

Column	Description
Unique SQL Id	ID of the normalized SQL statement.
Node Name	Node name.
User Name	Username.
Tuples Read	Number of tuples that are read.
Calls	Number of calls.
Min Elapse Time(us)	Minimum execution time (unit: us).
Max Elapse Time(us)	Maximum execution time (unit: us).
Total Elapse Time(us)	Total execution time (unit: us).
Avg Elapse Time(us)	Average execution time (unit: us).
Returned Rows	Number of rows returned by SELECT.
Tuples Affected	Number of rows affected by INSERT, UPDATE, and DELETE.
Logical Read	Number of logical reads on the buffer.
Physical Read	Number of physical reads on the buffer.
CPU Time(us)	CPU time (unit: us).
Data IO Time(us)	Time spent on I/O (unit: us).
Sort Count	Number of sorting execution times.
Sort Time(us)	Sorting execution time (unit: us).
Sort Mem Used(KB)	Size of work memory used during sorting (unit: KB).
Sort Spill Count	Number of file writes when data is flushed to disks during sorting.
Sort Spill Size(KB)	File size used when data is flushed to disks during sorting (unit: KB).

Column	Description
Hash Count	Number of hashing execution times.
Hash Time(us)	Hashing execution time (unit: us).
Hash Mem Used(KB)	Size of work memory used during hashing (unit: KB).
Hash Spill Count	Number of file writes when data is flushed to disks during hashing.
Hash Spill Size(KB)	File size used when data is flushed to disks during hashing (unit: KB).
SQL Text	Normalized SQL character string.

14.2.4.11 Wait Events

The following table describes columns in the Wait Events report.

Table 14-184 Columns in the Wait Events report

Column	Description
Type	Wait events are classified as follows: ● STATUS ● LWLOCK_EVENT ● LOCK_EVENT ● IO_EVENT
Event	Name of a wait event.
Total Wait Time (us)	Total wait time (unit: us).
Waits	Total number of wait times.
Failed Waits	Number of wait failures.
Avg Wait Time (us)	Average wait time (unit: us).
Max Wait Time (us)	Maximum wait time (unit: us).

14.2.4.12 Cache IO Stats

The Cache IO Stats report consists of two tables, namely, **User table IO activity** and **User index IO activity**. Columns in the tables are described as follows:

User table IO activity

Table 14-185 Columns in the User table IO activity table

Column	Description
DB Name	Database name.
Schema Name	Schema name.
Table Name	Table name.
%Heap Blks Hit Ratio	Buffer pool hit ratio of the table.
Heap Blks Read	Number of disk blocks read from the table.
Heap Blks Hit	Number of cache hits in the table.
Idx Blks Read	Number of disk blocks read from all indexes on the table.
Idx Blks Hit	Number of cache hits in the table .
Toast Blks Read	Number of disk blocks read from the TOAST table (if any) in the table.
Toast Blks Hit	Number of buffer hits in the TOAST table (if any) in the table.
Tidx Blks Read	Number of disk blocks read from the TOAST table index (if any) in the table.
Tidx Blks Hit	Number of buffer hits in the TOAST table index (if any) in the table.

User index IO activity

Table 14-186 Columns in the User index IO activity table

Column	Description
DB Name	Database name.
Schema Name	Schema name.
Table Name	Table name.

Column	Description
Index Name	Index name.
%Idx Blks Hit Ratio	Index hit ratio.
Idx Blks Read	Number of disk blocks read from all indexes on the table.
Idx Blks Hit	Number of cache hits in the table.

14.2.4.13 Utility status

Utility status contains two tables: **Replication slot** and **Replication stat**. Columns in the tables are described as follows:

Replication slot

Table 14-187 Columns in the Replication slot report

Column	Description
Slot Name	Replication node name.
Slot Type	Type of the replication node.
DB Name	Name of the database on the replication node.
Active	Replication node status.
Xmin	Transaction ID of the replication node.
Restart Lsn	Xlog file information on the replication node.
Dummy Standby	Replication node as a dummy standby.

Replication stat

Table 14-188 Columns in the Replication stat report

Column	Description
Thread Id	PID of the thread.
Usesys Id	User system ID.
Username	Username.
Application Name	Application name.

Column	Description
Client Addr	Client address.
Client Hostname	Client host name.
Client Port	Port of the client.
Backend Start	Start time of an application.
State	Log replication status.
Sender Sent Location	Location where the sender sends logs.
Receiver Write Location	Location where the receiver writes logs.
Receiver Flush Location	Location where the receiver flushes logs.
Receiver Replay Location	Location where the receiver replays logs.
Sync Priority	Synchronization priority.
Sync State	Synchronization status.

14.2.4.14 Object stats

Object stats contains three tables: User Tables stats, User index stats, and Bad lock stats. Columns in the tables are described as follows:

User Tables stats

Table 14-189 Columns in the User Tables stats report

Column	Description
DB Name	Database name.
Schema	Schema name.
Relname	Relation name.
Seq Scan	Number of sequential scans initiated on this table.
Seq Tup Read	Number of live rows fetched by sequential scans.
Index Scan	Number of index scans initiated on the table.
Index Tup Fetch	Number of live rows fetched by index scans.
Tuple Insert	Number of rows inserted.

Column	Description
Tuple Update	Number of rows updated.
Tuple Delete	Number of rows deleted.
Tuple Hot Update	Number of rows HOT updated (with no separate index updated).
Live Tuple	Estimated number of live rows.
Dead Tuple	Estimated number of dead rows.
Last Vacuum	Last time at which this table was manually vacuumed (not counting **VACUUM FULL**).
Last Autovacuum	Last time at which this table was vacuumed by the autovacuum daemon.
Last Analyze	Last time at which this table was manually analyzed.
Last Autoanalyze	Last time at which this table was analyzed by the autovacuum daemon.
Vacuum Count	Number of times the table has been manually vacuumed (not counting **VACUUM FULL**).
Autovacuum Count	Number of times the table has been vacuumed by the autovacuum daemon.
Analyze Count	Number of times the table has been manually analyzed.
Autoanalyze Count	Number of times the table has been analyzed by the autovacuum daemon.

User index stats

Table 14-190 Columns in the User index stats report

Column	Description
DB Name	Database name.
Schema	Schema name.
Relname	Relation name.
Index Relname	Index name.
Index Scan	Number of index scans initiated on the index.
Index Tuple Read	Number of index entries returned by scans on the index.

Column	Description
Index Tuple Fetch	Number of live table rows fetched by simple index scans using the index.

Bad lock stats

Table 14-191 Columns in the Bad lock stats report

Column	Description
DB Id	OID of the database.
Tablespace Id	Tablespace OID.
Relfilenode	File object ID.
Fork Number	File type.
Error Count	Number of failures.
First Time	First occurrence time.
Last Time	Last occurrence time.

14.2.4.15 Configuration settings

The following table describes columns in the Configuration settings report.

Table 14-192 Columns in the Configuration settings report

Column	Description
Name	GUC name.
Abstract	GUC description.
Type	Data type.
Curent Value	Current value.
Min Value	Valid minimum value.
Max Value	Valid maximum value.
Category	GUC type.
Enum Values	All enumerated values.
Default Value	Default parameter value used upon the database startup.

Column	Description
Reset Value	Default parameter value used upon the database reset.

14.2.4.16 SQL Detail

The following table describes columns in the SQL Detail report.

Table 14-193 Columns in the SQL Detail report

Column	Description
Unique SQL Id	ID of the normalized SQL statement.
User Name	Username.
Node Name	Node name. This column is not displayed in node mode.
SQL Text	Normalized SQL text.

14.3 DBE_PLDEBUGGER Schema

DBE_PLDEBUGGER Schema system functions are used to debug stored procedures. This chapter describes the interfaces supported by DBE_PLDEBUGGER Schema. Only the administrator has the permission to execute these debugging interfaces, but does not have the permission to modify or create functions.

NOTICE

When a user is created in the function body, the plaintext password is returned when attach, next, continue, info_code, step, info_breakpoint, backtrace or finish is called. You are not advised to create a user in the function body.

The administrator can run the following command to grant the **gs_role_pldebugger** role and debugger permissions to a user:

```
GRANT gs_role_pldebugger to user;
```

Two clients are required to connect to the database. One client is responsible for executing the debugging interface as the debug end, and the other client is responsible for executing the debugging function to control the execution of stored procedures on the server. The following is an example.

- Prepare for debugging.

  Use PG_PROC to find the OID of the stored procedure to be debugged and execute **DBE_PLDEBUGGER.turn_on(oid)**. In this case, the client functions as the server.

```
openGauss=# CREATE OR REPLACE PROCEDURE test_debug ( IN  x INT)
AS
BEGIN
    INSERT INTO t1 (a) VALUES (x);
    DELETE FROM t1 WHERE a = x;
END;
/
CREATE PROCEDURE
openGauss=# SELECT OID FROM PG_PROC WHERE PRONAME='test_debug';
  oid
-------
 16389
(1 row)
openGauss=# SELECT * FROM DBE_PLDEBUGGER.turn_on(16389);
 nodename | port
----------+------
 datanode |    0
(1 row)
```

- Start debugging.

  When the server executes the stored procedure, the server hangs before the first SQL statement in the stored procedure and waits for the debugging message sent by the debug end. Debugging is supported only by directly executing a stored procedure and cannot be achieved by invoking an executed stored procedure through a trigger.

```
openGauss=# call test_debug(1);
```

  Start another client as the debug end and invoke **DBE_PLDEBUGGER.attach** to attach with the stored procedure for debugging based on the data returned by **turn_on**.

```
openGauss=# SELECT * FROM DBE_PLDEBUGGER.attach('datanode',0);
 funcoid |  funcname  | lineno |              query
---------+------------+--------+----------------------------------
   16389 | test_debug |      3 |   INSERT INTO t1 (a) VALUES (x);
(1 row)
```

  Execute the next statement on the client where the attach operation is performed.

```
openGauss=# SELECT * FROM DBE_PLDEBUGGER.next();
 funcoid |  funcname  | lineno |        query
---------+------------+--------+----------------------
   16389 | test_debug |      0 | [EXECUTION FINISHED]
(1 row)
```

  Execute the following variable operations on the client where the attach command is performed.

```
openGauss=# SELECT * FROM DBE_PLDEBUGGER.info_locals(); -- Print all variables.
 varname | vartype | value | package_name | isconst
---------+---------+-------+--------------+---------
 x       | int4    | 1     |              | f
(1 row)
openGauss=# SELECT * FROM DBE_PLDEBUGGER.set_var('x', 2); -- Assign a value to a variable.
 set_var
---------
 t
(1 row)
openGauss=# SELECT * FROM DBE_PLDEBUGGER.print_var('x'); -- Print a single variable.
 varname | vartype | value | package_name | isconst
---------+---------+-------+--------------+---------
 x       | int4    | 2     |              | f
(1 row)
```

  Directly execute the stored procedure that is being debugged.

```
openGauss=# SELECT * FROM DBE_PLDEBUGGER.continue();
 funcoid |  funcname  | lineno |        query
---------+------------+--------+----------------------
```

```
  16389 | test_debug |      0 | [EXECUTION FINISHED]
(1 row)
```

Exit the stored procedure that is being debugged and do not execute statements that have not been executed before.

```
openGauss=# SELECT * FROM DBE_PLDEBUGGER.abort();
 abort
-------
 t
(1 row)
```

View the code information on the client and identify the line number of the breakpoint that can be set.

```
openGauss=# SELECT * FROM DBE_PLDEBUGGER.info_code(16389);
 lineno |                           query                           | canbreak
--------+-----------------------------------------------------------+----------
        | CREATE OR REPLACE PROCEDURE public.test_debug( IN  x INT) | f
      1 | AS  DECLARE                                               | f
      2 | BEGIN                                                     | f
      3 |     INSERT INTO t1 (a) VALUES (x);                        | t
      4 |     DELETE FROM t1 WHERE a = x;                           | t
      5 | END;                                                      | f
      6 | /                                                         | f
(7 rows)
```

Sets a breakpoint.

```
openGauss=# SELECT * FROM DBE_PLDEBUGGER.add_breakpoint(16389,4);
 lineno |                           query                           | canbreak
--------+-----------------------------------------------------------+----------
        | CREATE OR REPLACE PROCEDURE public.test_debug( IN  x INT) | f
      1 | AS  DECLARE                                               | f
      2 | BEGIN                                                     | f
      3 |     INSERT INTO t1 (a) VALUES (x);                        | t
      4 |     DELETE FROM t1 WHERE a = x;                           | t
      5 | END;                                                      | f
      6 | /                                                         | f
(7 rows)
```

View the breakpoint information.

```
openGauss=# SELECT * FROM DBE_PLDEBUGGER.info_breakpoints();
 breakpointno | funcoid | lineno |              query              | enable
--------------+---------+--------+---------------------------------+--------
            0 |   16389 |      4 |     DELETE FROM t1 WHERE a = x; | t
(1 row)
```

Execute to the breakpoint.

```
openGauss=# SELECT * FROM DBE_PLDEBUGGER.continue();
 funcoid |  funcname  | lineno |              query
---------+------------+--------+--------------------------------
   16389 | test_debug |      4 |     DELETE FROM t1 WHERE a = x;
(1 row)
```

After the stored procedure is executed, the debugging automatically exits. To debug the stored procedure again, you need to attach again. If the server does not need to be debugged, run the **turn_off** command to disable the debugging or exit the session. For details about the debugging interfaces, see the following table.

Table 14-194 DBE_PLDEBUGGER

Interface	Description
DBE_PLDEBUGGER.turn_on	Invoked by the server, indicating that the stored procedure can be debugged. After the interface is invoked, the stored procedure is hung to wait for debugging information.
DBE_PLDEBUGGER.turn_off	Invoked by the server, indicating that debugging the stored procedure is disabled.
DBE_PLDEBUGGER.local_debug_server_info	Invoked by the server to print all stored procedures that have been turned on in the current session.
DBE_PLDEBUGGER.attach	Invoked by the debug end to attach with the stored procedure that is being debugged.
DBE_PLDEBUGGER.info_locals	Invoked by the debug end to print the current values of variables in the stored procedure that is being debugged.
DBE_PLDEBUGGER.next	Invoked by the debug end to execute the next step.
DBE_PLDEBUGGER.continue	Invoked by the debug end to continue the execution until the breakpoint or stored procedure ends.
DBE_PLDEBUGGER.abort	Invoked by the debug end to stop debugging. The server reports a long jump error.
DBE_PLDEBUGGER.print_var	Invoked by the debug end to print the current values of specified variables in the stored procedure that is being debugged.
DBE_PLDEBUGGER.info_code	Invoked by the debug end or server to print the source statement of a specified stored procedure and the line number corresponding to each line.
DBE_PLDEBUGGER.step	Invoked by the debug end to execute step by step.
DBE_PLDEBUGGER.add_breakpoint	Invoked by the debug end to add a breakpoint.
DBE_PLDEBUGGER.delete_breakpoint	Invoked by the debug end to delete a breakpoint.
DBE_PLDEBUGGER.info_breakpoints	Invoked by the debug end to view all breakpoints.
DBE_PLDEBUGGER.backtrace	Invoked by the debug end to check the current call stack.
DBE_PLDEBUGGER.enable_breakpoint	Invoked by the debug end to enable breakpoints.

Interface	Description
DBE_PLDEBUGGER.disable_breakpoint	Invoked by the debug end to disable breakpoints.
DBE_PLDEBUGGER.finish	Invoked by the debug end to continue the debugging until the breakpoint is reached or the upper-layer call stack is returned.
DBE_PLDEBUGGER.set_var	Invoked by the debug end to assign a value to a variable.

14.3.1 DBE_PLDEBUGGER.turn_on

This function is used to mark a stored procedure as debuggable. After **turn_on** is executed, the server can execute the stored procedure for debugging. You need to manually obtain the OID of the stored procedure based on the PG_PROC system catalog and transfer it to the function. After **turn_on** is executed, the execution of the stored procedure in the current session is hung before the first SQL statement to wait for the debugging instruction from the debug end. This setting is cleared by default after the session is disconnected. Currently, stored procedures and functions with autonomous transactions enabled cannot be debugged.

The function prototype is as follows:

```
DBE_PLDEBUGGER.turn_on(Oid)
RETURN Record;
```

Table 14-195 turn_on input parameters and return values

Name	Type	Description
func_oid	IN oid	Function OID
nodename	OUT text	Node name
port	OUT integer	Number of the connected port

14.3.2 DBE_PLDEBUGGER.turn_off

This function is used to remove the debugging flag added by **turn_on**. The return value indicates success or failure. You can run the **DBE_PLDEBUGGER.local_debug_server_info** command to query the OID of the stored procedure that has been turned on.

The function prototype is as follows:

```
DBE_PLDEBUGGER.turn_off(Oid)
RETURN boolean;
```

Table 14-196 turn_off input parameters and return values

Name	Type	Description
func_oid	IN oid	Function OID
turn_off	OUT boolean	Whether turn-off is successful

14.3.3 DBE_PLDEBUGGER.local_debug_server_info

This function is used to query the OID of the stored procedure that has been turned on in the current connection. You can use **funcoid** and **pg_proc** together to determine which stored procedures are to be debugged.

Table 14-197 local_debug_server_info returned input parameters and return values

Name	Type	Description
nodename	OUT text	Node name
port	OUT bigint	Port number
funcoid	OUT oid	Stored procedure OID

14.3.4 DBE_PLDEBUGGER.attach

When the server executes a stored procedure, the server hangs the execution before the first statement and waits for attaching with the debug end. The debug end invokes the attach function and transfers node name and port number to attach with the specified stored procedure.

If an error is reported during debugging, the attach operation automatically becomes invalid. If the debug end is attached to another stored procedure during debugging, the debugging of the attached stored procedure becomes invalid.

Table 14-198 attach input parameters and return values

Name	Type	Description
nodename	IN text	Node name
port	IN integer	Number of the connected port
funcoid	OUT oid	Function ID
funcname	OUT text	Function name

Name	Type	Description
lineno	OUT integer	Number of the next line in the current debugging process
query	OUT text	Source code of the next line of the function that is being debugged

14.3.5 DBE_PLDEBUGGER.info_locals

During debugging on the debug end, **info_locals** is invoked to print the variables in the current stored procedure. The input parameter **frameno** of this function indicates the stack layer to be traversed. This function can be invoked without input parameters. By default, the top-layer stack variable is queried.

Table 14-199 info_locals input parameters and return values

Name	Type	Description
frameno	IN integer (optional)	Specified stack layer. The default value is the top layer.
varname	OUT text	Variable name
vartype	OUT text	Variable type
value	OUT text	Variable value
package_name	OUT text	Name of the package corresponding to the variable. If the variable is not a package, the value is null.
isconst	OUT boolean	Whether it is a constant

14.3.6 DBE_PLDEBUGGER.next

This function is used to execute the current SQL statement in a stored procedure and return the number of the next SQL statement and the corresponding query.

Table 14-200 next input parameters and return values

Name	Type	Description
funcoid	OUT oid	Function ID

Name	Type	Description
funcname	OUT text	Function name
lineno	OUT integer	Number of the next line in the current debugging process
query	OUT text	Source code of the next line of the function that is being debugged

14.3.7 DBE_PLDEBUGGER.continue

Executes the current stored procedure until reaching the next breakpoint or end, and returns the line number of the next execution and the corresponding query.

The function prototype is as follows:

```
DBE_PLDEBUGGER.continue()
RETURN Record;
```

Table 14-201 continue input parameters and return values

Name	Type	Description
funcoid	OUT oid	Function ID
funcname	OUT text	Function name
lineno	OUT integer	Number of the next line in the current debugging process
query	OUT text	Source code of the next line of the function that is being debugged

14.3.8 DBE_PLDEBUGGER.abort

This function is used to abort the stored procedure executed on the server and report an error. The return value indicates whether the abort message is successfully sent.

The function prototype is as follows:

```
DBE_PLDEBUGGER.abort()
RETURN boolean;
```

Table 14-202 abort input parameters and return values

Name	Type	Description
abort	OUT boolean	Success or failure

14.3.9 DBE_PLDEBUGGER.print_var

During debugging on the debug end, **print_var** is invoked to print the name and value of the specified variable in the current stored procedure. The input parameter **frameno** of this function indicates the stack layer to be traversed. This function can be invoked without **frameno**. By default, the top-layer stack variable is queried.

Table 14-203 print_var input parameters and return values

Name	Type	Description
var_name	IN text	Variable
frameno	IN integer (optional)	Specified stack layer. The default value is the top layer.
varname	OUT text	Variable name
vartype	OUT text	Variable type
value	OUT text	Variable value
package_name	OUT text	Package name corresponding to the variable. This parameter is reserved and is left empty currently.
isconst	OUT boolean	Whether it is a constant

14.3.10 DBE_PLDEBUGGER.info_code

During debugging on the debug end, **info_code** is invoked to view the source statement of the specified stored procedure and the line number corresponding to each line. The line number starts from the function body, and the line number in the function header is empty.

Table 14-204 info_code input parameters and return values

Name	Type	Description
funcoid	IN oid	Function ID

Name	Type	Description
lineno	OUT integer	Line number
query	OUT text	Source statement
canbreak	OUT bool	Specifies whether the current line supports breakpoints.

14.3.11 DBE_PLDEBUGGER.step

During debugging on the debug end, if a stored procedure is being executed, the stored procedure continues to be executed and information such as the line number in the first line of the stored procedure is returned. If the executed object is not a stored procedure, the return is the same as that for **next**. After the SQL statement is executed, information such as the line number in the next line is returned.

Table 14-205 step input parameters and return values

Name	Type	Description
funcoid	OUT oid	Function ID
funcname	OUT text	Function name
lineno	OUT integer	Number of the next line in the current debugging process
query	OUT text	Source code of the next line of the function that is being debugged

14.3.12 DBE_PLDEBUGGER.add_breakpoint

During debugging on the debug end, call **add_breakpoint** to add a breakpoint. If **–1** is returned, the specified breakpoint is invalid. Determine the proper position of the breakpoint based on the **canbreak** column in **DBE_PLDEBUGGER.info_code**.

Table 14-206 add_breakpoint input parameters and return values

Name	Type	Description
funcoid	IN text	Function ID
lineno	IN integer	Line number
breakpointno	OUT integer	Breakpoint number

14.3.13 DBE_PLDEBUGGER.delete_breakpoint

During debugging on the debug end, call **delete_breakpoint** to delete the existing breakpoint.

Table 14-207 delete_breakpoint input parameters and return values

Name	Type	Description
breakpointno	IN integer	Breakpoint number
result	OUT bool	Specifies whether this operation is successful.

14.3.14 DBE_PLDEBUGGER.info_breakpoints

During debugging on the debug end, call **info_breakpoints** to view the current function breakpoint.

Table 14-208 info_breakpoints input parameters and return values

Name	Type	Description
breakpointno	OUT integer	Breakpoint number
funcoid	OUT oid	Function ID
lineno	OUT integer	Line number
query	OUT text	Breakpoint content
enable	OUT boolean	Valid or not

14.3.15 DBE_PLDEBUGGER.backtrace

During debugging on the debug end, call **backtrace** to view the current call stack.

Table 14-209 backtrace input parameters and return values

Name	Type	Description
frameno	OUT integer	Call stack ID
funcname	OUT text	Function name
lineno	OUT integer	Line number
query	OUT text	Breakpoint content
funcoid	OUT oid	Function OID

14.3.16 DBE_PLDEBUGGER.enable_breakpoint

During debugging on the debug end, calls enable_breakpoint to enable breakpoints.

Table 14-210 enable_breakpoint input parameters and return values

Name	Type	Description
breakpointno	IN integer	Breakpoint number
result	OUT bool	Whether this operation is successful

14.3.17 DBE_PLDEBUGGER.disable_breakpoint

During debugging on the debug end, call disable_breakpoint to disable breakpoints.

Table 14-211 disable_breakpoint input parameters and return values

Name	Type	Description
breakpointno	IN integer	Breakpoint number
result	OUT bool	Whether this operation is successful

14.3.18 DBE_PLDEBUGGER.finish

Executes the current SQL statement in the stored procedure until the next breakpoint is triggered or the next line of the upper-layer stack is executed.

Table 14-212 finish input parameters and return values

Name	Type	Description
funcoid	OUT oid	Function ID
funcname	OUT text	Function name
lineno	OUT integer	Number of the next line in the current debugging process
query	OUT text	Source code of the next line of the function that is being debugged

14.3.19 DBE_PLDEBUGGER.set_var

Changes the variable on the top-layer stack in the specified debugging stored procedure to the value of the input parameter. If a stored procedure contains variables with the same name, set_var supports only the setting of the first variable value.

Table 14-213 set_var input parameters and return values

Name	Type	Description
var_name	IN text	Variable name
value	IN text	New value
result	OUT boolean	Result (success or failure)

14.4 DBE_SQL_UTIL Schema

The DBE_SQL_UTIL schema stores tools for managing SQL patches, including creating, deleting, enabling, and disabling SQL patches. Common users have only the USAGE permission and do not have the CREATE, ALTER, DROP, and COMMENT permissions.

14.4.1 DBE_SQL_UTIL.create_hint_sql_patch

create_hint_sql_patch is an interface function used to create SQL patches for hints. It returns whether the execution is successful.

Only the initial user, sysadmin, opradmin, and monadmin have the permission to invoke this interface.

Table 14-214 DBE_SQL_UTIL.create_hint_sql_patch input parameters and return values

Parameter	Type	Description
patch_name	IN name	Patch name.
unique_sql_id	IN bigint	Global unique ID.
hint_string	IN text	Hint text.
description	IN text	Patch description. The default value is **NULL**.
enabled	IN bool	Specifies whether the patch takes effect. The default value is **true**.

14.4.2 DBE_SQL_UTIL.create_abort_sql_patch

create_abort_sql_patch is an interface function used to create abort SQL patches. It returns whether the execution is successful.

Only the initial user, sysadmin, opradmin, and monadmin have the permission to invoke this interface.

Table 14-215 DBE_SQL_UTIL.create_abort_sql_patch input parameters and return values

Parameter	Type	Description
patch_name	IN name	Patch name.
unique_sql_id	IN bigint	Global unique ID.
description	IN text	Patch description. The default value is **NULL**.
enabled	IN bool	Specifies whether the patch takes effect. The default value is **true**.
result	OUT bool	Specifies whether this operation is successful.

14.4.3 DBE_SQL_UTIL.drop_sql_patch

drop_sql_patch is an interface function used to delete SQL patches. It returns whether the execution is successful.

Only the initial user, sysadmin, opradmin, and monadmin have the permission to invoke this interface.

Table 14-216 DBE_SQL_UTIL.drop_sql_patch input parameters and return values

Parameter	Type	Description
patch_name	IN name	Patch name.
result	OUT bool	Specifies whether this operation is successful.

14.4.4 DBE_SQL_UTIL.enable_sql_patch

enable_sql_patch is an interface function used to enable SQL patches. It returns whether the execution is successful.

Only the initial user, sysadmin, opradmin, and monadmin have the permission to invoke this interface.

Table 14-217 DBE_SQL_UTIL.enable_sql_patch input parameters and return values

Parameter	Type	Description
patch_name	IN name	Patch name.
result	OUT bool	Specifies whether this operation is successful.

14.4.5 DBE_SQL_UTIL.disable_sql_patch

disable_sql_patch is an interface function used to disable SQL patches. It returns whether the execution is successful.

Only the initial user, sysadmin, opradmin, and monadmin have the permission to invoke this interface.

Table 14-218 DBE_SQL_UTIL.disable_sql_patch input parameters and return values

Parameter	Type	Description
patch_name	IN name	Patch name.
result	OUT bool	Specifies whether this operation is successful.

14.4.6 DBE_SQL_UTIL.show_sql_patch

show_sql_patch is an interface function used to display the SQL patch corresponding to a specified patch name and return the running result.

Only the initial user, sysadmin, opradmin, and monadmin have the permission to invoke this interface.

Table 14-219 DBE_SQL_UTIL.show_sql_patch input parameters and return values

Parameter	Type	Description
patch_name	IN name	Patch name.
unique_sql_id	OUT bigint	Global unique ID.
enabled	OUT bool	Specifies whether the patch takes effect.
abort	OUT bool	Specifies whether the patch is an abort hint.
hint_str	OUT text	Hint text.

15 Logical Replication

15.1 Logical Decoding

15.1.1 Overview

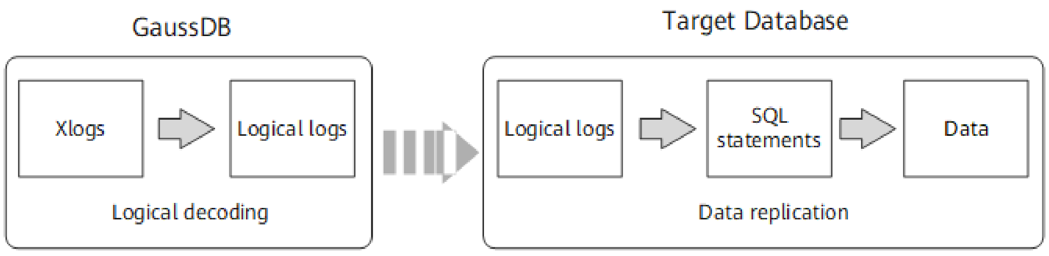
Function

The data replication capabilities supported by GaussDB are as follows:

Data is periodically synchronized to heterogeneous databases using a data migration tool. Real-time data replication is not supported. Therefore, the requirements for real-time data synchronization to heterogeneous databases are not satisfied.

GaussDB provides the logical decoding function to generate logical logs by decoding Xlogs. A target database parses logical logs to replicate data in real time. For details, see **Figure 15-1**. Logical replication reduces the restrictions on target databases, allowing for data synchronization between heterogeneous databases and homogeneous databases with different forms. It allows data to be read and written during data synchronization on a target database, reducing the data synchronization latency.

Figure 15-1 Logical replication



Logical replication consists of logical decoding and data replication. Logical decoding outputs logical logs by transaction. The database service or middleware parses the logical logs to implement data replication. Currently, GaussDB supports only logical decoding. Therefore, this section involves only logical decoding.

Logical decoding provides basic transaction decoding capabilities for logical replication. GaussDB uses SQL functions for logical decoding. This method features easy function calling, requires no tools to obtain logical logs, and provides specific interfaces for interconnecting with external replay tools, saving the need of additional adaptation.

Logical logs are output only after transactions are committed because they use transactions as the unit and logical decoding is driven by users. Therefore, to prevent Xlogs from being recycled by the system when transactions start and prevent required transaction information from being recycled by **VACUUM**, GaussDB introduces logical replication slots to block Xlog recycling.

A logical replication slot means a stream of changes that can be replayed in other databases in the order they were generated in the original database. Each owner of logical logs maintains one logical replication slot.

Precautions

- DDL statement decoding is not supported. When a specific DDL statement (for example, to truncate an ordinary table or exchange a partitioned table) is executed, decoded data may be lost.
- Decoding for column-store data and data page replication is not supported.
- After a DDL statement (for example, **ALTER TABLE**) is executed, the physical logs that are not decoded before the DDL statement execution may be lost.
- The size of a single tuple cannot exceed 1 GB, and decoded data may be larger than inserted data. Therefore, it is recommended that the size of a single tuple be less than or equal to 500 MB.
- GaussDB supports the following data types for decoding: **INTEGER**, **BIGINT**, **SMALLINT**, **TINYINT**, **SERIAL**, **SMALLSERIAL**, **BIGSERIAL**, **FLOAT**, **DOUBLE PRECISION**, **DATE**, **TIME[WITHOUT TIME ZONE]**, **TIMESTAMP[WITHOUT TIME ZONE]**, **CHAR(*n*)**, **VARCHAR(*n*)**, and **TEXT**.
- If the SSL connection is required, ensure that the GUC parameter **ssl** is set to **on**.
- The logical replication slot name must contain fewer than 64 characters and contain only one or more types of the following characters: lowercase letters, digits, and underscores (_).
- After the database where a logical replication slot resides is deleted, the replication slot becomes unavailable and needs to be manually deleted.
- To decode multiple databases, you need to create a stream replication slot in each database and start decoding. Logs need to be scanned for decoding of each database.
- Forcible switchover is not supported. After forcible switchover, you need to export all data again.
- To perform decoding on the standby node, set the GUC parameter **enable_slot_log** to **on** on the corresponding host.
- During decoding on the standby node, the decoded data may increase during switchover and failover, which needs to be manually filtered out. When the quorum protocol is used, switchover and failover should be performed on the standby node that is to be promoted to primary, and logs must be synchronized from the primary node to the standby node.

- The same replication slot for decoding cannot be used between the primary node and standby node or between different standby nodes at the same time. Otherwise, data inconsistency occurs.
- Replication slots can only be created or deleted on hosts.
- After the database is restarted due to a fault or the logical replication process is restarted, duplicate decoded data may exist. You need to filter out the duplicate data.
- If the computer kernel is faulty, garbled characters may be displayed during decoding, which need to be manually or automatically filtered out.
- Currently, the logical decoding on the standby node does not support enabling the ultimate RTO.
- Ensure that the long transaction is not started during the creation of the logical replication slot. If the long transaction is started, the creation of the logical replication slot will be blocked.
- Interval partitioned tables cannot be replicated.
- Global temporary tables are not supported.
- After a DDL statement is executed in a transaction, the DDL statement and subsequent statements are not decoded.
- Do not perform operations on the replication slot on other nodes when the logical replication slot is in use. To delete a replication slot, stop decoding in the replication slot first.
- To parse the UPDATE and DELETE statements of an Astore table, you need to configure the **REPLICA IDENITY** attribute for the table. If the table does not have a primary key, set the **REPLICA IDENITY** attribute to **FULL**. For details, see **▪REPLICA IDENTITY { DEFA...**.
- Do not perform operations on the replication slot on other nodes when the logical replication slot is in use. To delete a replication slot, stop decoding in the replication slot first.
- Considering that the target database may require the system status information of the source database, logical decoding automatically filters only logical logs of system catalogs whose OIDs are less than 16384 in **pg_catalog** and **pg_toast** schemas. If the target database does not need to copy the content of other related system catalogs, the related system catalogs need to be filtered during logical log replay.
- When logical replication is enabled, if you need to create a primary key index that contains system columns, you must set the **REPLICA IDENTITY** attribute of the table to **FULL** or use USING INDEX to specify a unique, non-local, non-deferrable index that does not contain system columns and contains only columns marked **NOT NULL**.

Performance

When **pg_logical_slot_get_changes** is used in the BenchmarkSQL 5.0 with 100 warehouses:

- If 4000 lines of data (about 5 MB to 10 MB logs) are decoded at a time, the decoding performance ranges from 0.3 MB/s to 0.5 MB/s.
- If 32000 lines of data (about 40 MB to 80 MB logs) are decoded at a time, the decoding performance ranges from 3 MB/s to 5 MB/s.

- If 256000 lines of data (about 320 MB to 640 MB logs) are decoded at a time, the decoding performance ranges from 3 MB/s to 5 MB/s.
- If the amount of data to be decoded at a time still increases, the decoding performance is not significantly improved.

If **pg_logical_slot_peek_changes** and **pg_replication_slot_advance** are used, the decoding performance is 30% to 50% lower than that when **pg_logical_slot_get_changes** is used.

15.1.2 Logical Decoding by SQL Function Interfaces

In GaussDB, you can call SQL functions to create, delete, and push logical replication slots, as well as obtain decoded transaction logs.

Prerequisites

- Currently, logical logs are extracted from host nodes. To perform logical replication, the GUC parameter **ssl** must be set to **on** on host nodes.

  NOTE

  For security purposes, ensure that SSL connections are enabled.
- The GUC parameter **wal_level** is set to **logical**.
- The GUC parameter **max_replication_slots** is set to a value greater than or equal to the number of physical streaming replication slots plus backup slots and logical replication slots required by each node.

  Physical streaming replication slots provide an automatic method to ensure that Xlogs are not removed from a primary node before they are received by all the standby nodes and secondary nodes. That is, physical replication slots are used to support primary/standby HA. The number of physical replication slots required by a database is equal to the ratio of standby and secondary nodes to the primary node. If an HA cluster has one primary node and three standby nodes, the number of required physical replication slots will be 3.

  Plan the number of logical replication slots as follows:
  - A logical replication slot can carry changes of only one database for decoding. If multiple databases are involved, create multiple logical replication slots.
  - If logical replication is needed by multiple target databases, create multiple logical replication slots in the source database. Each logical replication slot corresponds to one logical replication link.
- Only initial users and users with the **REPLICATION** permission can perform this operation. When separation of duties is disabled, database administrators can perform logical replication operations. When separation of duties is enabled, database administrators are not allowed to perform logical replication operations.
- Currently, primary/standby/secondary deployment is not supported by default.

Procedure

Step 1 Log in to the primary node of the GaussDB database as the database installation user.

Step 2 Run the following command to connect to the default database **postgres**:

```
gsql -d postgres -p 16000 -r
```

In this command, **16000** is the database port number. It can be replaced by an actual port number.

Step 3 Create a logical replication slot named **slot1**.

```
openGauss=# SELECT * FROM pg_create_logical_replication_slot('slot1', 'mppdb_decoding');
slotname | xlog_position
----------+---------------
slot1    | 0/601C150
(1 row)
```

Step 4 Create a table **t** in the database and insert data into it.

```
openGauss=# CREATE TABLE t(a int PRIMARY KEY, b int);
openGauss=# INSERT INTO t VALUES(3,3);
```

Step 5 Read the decoding result of **slot1**. The number of decoded records is 4096.

```
openGauss=# SELECT * FROM pg_logical_slot_peek_changes('slot1', NULL, 4096);
location  |  xid  | data
-----------+-------
+----------------------------------------------------------------------------------------------------------------------------
---------------------
-------------------------------------------
 0/601C188 | 1010023 | BEGIN 1010023
 0/601ED60 | 1010023 | COMMIT 1010023 CSN 1010022
 0/601ED60 | 1010024 | BEGIN 1010024
 0/601ED60 | 1010024 | {"table_name":"public.t","op_type":"INSERT","columns_name":
["a","b"],"columns_type":["integer","integer"],"columns_val":["3","3"],"old_keys_name":[],"old_keys_type":
[],"old_keys_val":[]}
 0/601EED8 | 1010024 | COMMIT 1010024 CSN 1010023
(5 rows)
```

Step 6 Delete the logical replication slot **slot1**.

```
openGauss=#  SELECT * FROM pg_drop_replication_slot('slot1');
 pg_drop_replication_slot
--------------------------

(1 row)
```

----End

15.2 Replicating Data Using the Logical Replication Tool

Currently, SDR and DRS support GaussDB logical replication. The replication tool extracts logical logs from GaussDB and replays them in the peer database. For details about the code of the replication tool that uses JDBC to connect to the database, see **Example: Logic Replication Code**.

15.3 Publication-Subscription

Publication-subscription is implemented based on logical replication, with one or more subscribers subscribing to one or more publications on a publisher node. The subscriber pulls data from the publication they subscribe to.

Changes on the publisher are sent to the subscriber as they occur in real time. The subscriber applies the data in the same order as the publisher, so that transactional consistency is guaranteed for publications within a single subscription. This method of data replication is sometimes called transactional replication.

The typical usage of publication-subscription is as follows:

- Sending incremental changes in a database or a subset of a database to subscribers as they occur
- Firing triggers when changes reach subscribers
- Consolidating multiple databases into a single one (for example, for analysis purposes)

The subscriber database behaves in the same way as any other GaussDB instance and can be used as a publisher for other databases by defining its own publications. When the subscriber is treated as read-only by an application, there will be no conflicts in a single subscription. On the other side, conflicts may occur if other write operations are performed by the application or by other subscribers in the same set of tables.

15.3.1 Publications

A publication can be defined on any primary physical replication server. The node where a publication is defined is called the publisher. A publication is a set of changes generated from a table or a group of tables. It can also be described as a change set or replication set. Each publication exists in only one database.

Publications are different from schemas and do not affect how tables are accessed. Each table can be added to multiple publications if needed. Currently, publications can contain only tables. Objects must be explicitly added to a publication, unless the publication is created by running **ALL TABLES**.

Publications can choose to limit the changes they produce to any combination of **INSERT**, **UPDATE**, and **DELETE**, which is similar to how triggers are fired by particular events. By default, all types of operations are replicated.

A published table must be configured with a "replication identifier" in order to be able to replicate UPDATE and DELETE operations, so that appropriate rows to be updated or deleted can be identified on the subscriber side. By default, the replication identifier is the primary key (if any). You can also set another unique index (with certain additional requirements) to be the replication identifier. If the table does not have any suitable key, you can set the replication identifier to "full", which indicates that the entire row becomes the key. However, this is very inefficient and should be used only when there are no other solutions. If a replication identifier other than "full" is set on the publisher side, a replication identifier comprising the same or fewer columns must also be set on the subscriber side. For details about how to set a replication identifier, see **REPLICA IDENTITY** in **ALTER TABLE**. If a table without a replication identifier is added to a publication that replicates UPDATE or DELETE operations, subsequent UPDATE or DELETE operations on the subscriber side will cause an error. INSERT operations can proceed regardless of any replication identifier.

Each publication can have multiple subscribers.

A publication is created by running the **CREATE PUBLICATION** command and can be altered or dropped by running the corresponding commands.

Tables can be added or removed dynamically by running the **ALTER PUBLICATION** command. ADD TABLE and DROP TABLE operations are transactional. Therefore, once the transaction is committed, table replication will start or stop using a proper snapshot.

15.3.2 Subscriptions

A subscription is the downstream side of logical replication. The node where a subscription is defined is called the subscriber. A subscription defines the connection to another database and the set of publications (one or more) that it wants to subscribe to.

The subscriber database behaves in the same way as any other GaussDB instance and can be used as a publisher for other databases by defining its own publications.

A subscriber node can have multiple subscriptions if needed. You can define multiple subscriptions between a pair of publishers and subscribers, in which case you need to ensure that the published objects do not overlap.

Each subscription will receive changes through a replication slot. Currently, initial data in pre-existing tables cannot be synchronized.

If the current user is a user with the SYSADMIN permission, subscriptions are dumped by **pg_dump**. Otherwise, the subscriptions are skipped and a warning is written because users without the SYSADMIN permission cannot read all subscription information from the **pg_subscription** directory.

You can use **CREATE SUBSCRIPTION** to add a subscription, **ALTER SUBSCRIPTION** to alter a subscription, and **DROP SUBSCRIPTION** to drop a subscription.

When a subscription is dropped and recreated, the synchronized information is lost. This means that the data must be resynchronized.

The schema definitions are not replicated, and the published tables must exist on the subscriber. Only regular tables can be replicated. For example, a view cannot be replicated.

The tables are matched between the publisher and the subscriber using fully qualified table names. Replication to differently-named tables on the subscriber is not supported.

Columns of a table are also matched by name. The order of columns in the subscribed table does not need to be the same as that in the published table. The data types of the columns do not need to be the same, as long as the text representation of the data can be converted to the target type. For example, you can replicate from a column of the integer type to a column of the bigint type. The target table can also have additional columns that do not exist in the published table. The additional columns will be filled with the default values specified in the definition of the target table.

15.3.3 Conflicts

Logical replication behaves similarly to common DML operations. Even if the data is modified locally on the subscriber node, logical replication updates the data based on the received changes. If the incoming data violates any constraints, the replication will stop. This situation is called a conflict. When UPDATE or DELETE operations are replicated, the missing data will not cause conflicts and such operations will be simply skipped.

A conflict will cause errors and stop the replication, which must be resolved manually by the user. Details about the conflict can be found in the subscriber's server log.

The conflict can be resolved either by changing the data on the subscriber (so that the data does not conflict with incoming data) or by skipping the transaction that conflicts with the existing data. The transaction can be skipped by calling the **pg_replication_origin_advance()** function with **node_name** corresponding to the subscription name, and an Xlog LSN. The current position of the replication source can be seen in the **pg_replication_origin_status system** view.

15.3.4 Restrictions

Publication-subscription is implemented based on logical replication and inherits all restrictions of logical replication. In addition, publication-subscription has the following additional restrictions or missing functions.

- Database schemas and DDL commands are not replicated. Initial schemas can be manually copied by using **gs_dump --schema-only**. Subsequent schema changes need to be manually synchronized.
- Sequence data is not replicated. The data in the serial or identifier columns backed by sequences will be replicated as part of the table, but the sequences themselves on the subscriber have no change. If the subscriber is a read-only database, this is not a problem. However, if some kind of switchover or failover to the subscriber is intended, the sequence needs to be updated to the latest value, either by copying the current data from the publisher (perhaps using **gs_dump**) or by determining a sufficiently large value from the tables themselves.
- Only tables, including partitioned tables, can be replicated. Attempts to replicate other types of relations, such as views, materialized views, or foreign tables, will result in errors.
- Multiple subscriptions in the same database cannot subscribe to the same publication (that is, the same published table). Otherwise, duplicate data or primary key conflicts may occur.
- If a published table contains data types that do not support B-tree or hash indexes (such as the geography types), the table must have a primary key so that UPDATE and DELETE operations can be successfully replicated to the subscriber. Otherwise, the replication will fail, and the message "FATAL: could not identify an equality operator for type xx" will be displayed on the subscriber.

15.3.5 Architecture

Changes on publishers are sent to subscribers in real time as they occur. The subscriber applies data in the order in which it is committed on the publisher to ensure transactional consistency of publications in any single subscription.

Logical replication is built with an architecture similar to physical streaming replication. It is implemented by the walsender and apply processes. The walsender process starts logical decoding of the WAL and loads the standard logical decoding plug-in (pgoutput). The plug-in transforms the changes read from the WAL into a logical replication protocol and filters the data according to the publication specifications. The data is then continuously transferred to the apply worker using the streaming replication protocol, and the apply worker maps the data to the local table and applies the changes they receive in the correct transactional order.

The apply process in the subscriber database always runs with **session_replication_role** set to **replica**, which produces the usual effects on triggers and constraints.

The logical replication apply process currently fires only row triggers, not statement triggers. However, the initial table synchronization is implemented through methods similar to **COPY** command execution, and therefore, row and statement triggers for INSERT are fired.

15.3.6 Monitoring

Because logical replication is based on an architecture similar to physical streaming replication, monitoring on a publication node is similar to monitoring on a primary physical replication node.

Monitoring information about subscriptions is available in the **pg_stat_subscription** view. This view contains one row for every subscription worker. A subscription can have zero or more active subscription workers depending on its state.

Normally, a single apply process runs for an enabled subscription. A disabled or crashed subscription does not have rows in this view. If data synchronization of any table is in progress, there will be additional workers for the tables being synchronized.

15.3.7 Security

The role used for the replication connection must have the **REPLICATION** attribute (or be a user with the **SYSADMIN** permission). If the role lacks **SUPERUSER** and **BYPASSRLS**, the publisher's row security policies can be executed. The access permission of the role must be configured in the **pg_hba.conf** file and the role must have the **LOGIN** attribute.

To create a publication, the user must have the **CREATE** permission on the database.

To add tables to a publication, the user must have ownership of the table. To create a publication that automatically publishes all tables, the user must be a user with **SYSADMIN** permission.

To create a subscription, the user must be a user with the **SYSADMIN** permission.

The subscription apply process will run in the local database with the privileges of a user with the **SYSADMIN** permission.

Privileges are only checked once at the start of the replication connection. They are not re-checked when each change record is read from the publisher and when each change is applied.

15.3.8 Configuration Settings

Publication-subscription requires some configuration options to be set.

- On the publisher, **wal_level** must be set to **logical**, and the value of **max_replication_slots** must be at least the minimum number of subscriptions expected to be connected. **max_wal_senders** should be set to at least the value of **max_replication_slots** plus the number of physical replicas that are connected at the same time.
- **max_replication_slots** must also be set on the subscriber. It must be set to at least the number of subscriptions that will be added to the subscriber. **max_logical_replication_workers** must be set to at least the minimum number of subscriptions.

15.3.9 Quick Setup

Set the following configuration items in the **postgresql.conf** file:

```
wal_level = logical
```

For a basic setup, retain the default values for the other necessary configuration items.

You need to adjust the **pg_hba.conf** file to allow replication (the value depends on the actual network configuration and the user used for connection).

```
host     all     repuser     0.0.0.0/0     sha256
```

In the publisher database:

```
CREATE PUBLICATION mypub FOR TABLE users, departments;
```

In the subscriber database:

```
CREATE SUBSCRIPTION mysub CONNECTION 'dbname=foo host=bar user=repuser' PUBLICATION mypub;
```

The above statements start the replication process, replicating incremental changes to those tables.

16 Materialized View

A materialized view is a special physical table, which is relative to a common view. A common view is a virtual table and has many application limitations. Any query on a view is actually converted into a query on an SQL statement, and performance is not actually improved. The materialized view actually stores the results of the statements executed by the SQL statement, and is used to cache the results.

Currently, the Ustore engine does not support the creation and use of materialized views.

16.1 Complete-refresh Materialized View

16.1.1 Overview

Complete-refresh materialized views can be fully refreshed only. The syntax for creating a complete-refresh materialized view is similar to the CREATE TABLE AS syntax.

16.1.2 Usage

Syntax

- Create a complete-refresh materialized view.
  ```
  CREATE MATERIALIZED VIEW [ view_name ] AS { query_block };
  ```
- Fully refresh a materialized view.
  ```
  REFRESH MATERIALIZED VIEW [ view_name ];
  ```
- Delete a materialized view.
  ```
  DROP MATERIALIZED VIEW [ view_name ];
  ```
- Query a materialized view.
  ```
  SELECT * FROM [ view_name ];
  ```

Examples

```
-- Prepare data.
openGauss=# CREATE TABLE t1(c1 int, c2 int);
openGauss=# INSERT INTO t1 VALUES(1, 1);
openGauss=# INSERT INTO t1 VALUES(2, 2);
```

```
-- Create a complete-refresh materialized view.
openGauss=# CREATE MATERIALIZED VIEW mv AS select count(*) from t1;
CREATE MATERIALIZED VIEW

-- Query the materialized view result.
openGauss=# SELECT * FROM mv;
 count
-------
     2
(1 row)

-- Insert data into the base table in the materialized view.
openGauss=# INSERT INTO t1 VALUES(3, 3);
INSERT 0 1

-- Fully refresh the complete-refresh materialized view.
openGauss=# REFRESH MATERIALIZED VIEW mv;
REFRESH MATERIALIZED VIEW

-- Query the materialized view result.
openGauss=# SELECT * FROM mv;
 count
-------
     3
(1 row)

-- Delete the materialized view.
openGauss=# DROP MATERIALIZED VIEW mv;
DROP MATERIALIZED VIEW
```

16.1.3 Support and Constraints

Supported Scenarios

- Supports the same query scope as the CREATE TABLE AS statement does.
- Supports index creation in complete-refresh materialized views.
- Supports ANALYZE and EXPLAIN.

Unsupported Scenarios

Materialized views cannot be added, deleted, or modified. They support only query statements.

Constraints

When a complete-refresh materialized view is refreshed or deleted, a high-level lock is added to the base table. If the definition of a materialized view involves multiple tables, pay attention to the service logic to avoid deadlock.

16.2 Fast-refresh Materialized View

16.2.1 Overview

Fast-refresh materialized views can be incrementally refreshed. You need to manually execute statements to incrementally refresh materialized views in a period of time. The difference between the fast-refresh and the complete-refresh materialized views is that the fast-refresh materialized views support only a small number of scenarios. Currently, only base table scanning statements or UNION ALL can be used to create materialized views.

16.2.2 Usage

Syntax

- Create an fast-refresh materialized view.
  ```
  CREATE INCREMENTAL MATERIALIZED VIEW [ view_name ] AS { query_block };
  ```
- Fully refresh a materialized view.
  ```
  REFRESH MATERIALIZED VIEW [ view_name ];
  ```
- Incrementally refresh a materialized view.
  ```
  REFRESH INCREMENTAL MATERIALIZED VIEW [ view_name ];
  ```
- Delete a materialized view.
  ```
  DROP MATERIALIZED VIEW [ view_name ];
  ```
- Query a materialized view.
  ```
  SELECT * FROM [ view_name ];
  ```

Examples

```
-- Prepare data.
openGauss=# CREATE TABLE t1(c1 int, c2 int);
openGauss=# INSERT INTO t1 VALUES(1, 1);
openGauss=# INSERT INTO t1 VALUES(2, 2);

-- Create an fast-refresh materialized view.
openGauss=# CREATE INCREMENTAL MATERIALIZED VIEW mv AS SELECT * FROM t1;
CREATE MATERIALIZED VIEW

-- Insert data.
openGauss=# INSERT INTO t1 VALUES(3, 3);
INSERT 0 1

-- Incrementally refresh the materialized view.
openGauss=# REFRESH INCREMENTAL MATERIALIZED VIEW mv;
REFRESH MATERIALIZED VIEW

-- Query the materialized view result.
openGauss=# SELECT * FROM mv;
 c1 | c2
----+----
  1 |  1
  2 |  2
  3 |  3
(3 rows)

-- Insert data.
openGauss=# INSERT INTO t1 VALUES(4, 4);
INSERT 0 1

-- Fully refresh the materialized view.
openGauss=# REFRESH MATERIALIZED VIEW mv;
REFRESH MATERIALIZED VIEW

-- Query the materialized view result.
openGauss=# select * from mv;
 c1 | c2
----+----
  1 |  1
  2 |  2
  3 |  3
  4 |  4
(4 rows)
```

```
-- Delete the materialized view.
openGauss=# DROP MATERIALIZED VIEW mv;
DROP MATERIALIZED VIEW
```

16.2.3 Support and Constraints

Supported Scenarios

- Supports statements for querying a single table.
- Supports UNION ALL for querying multiple single tables.
- Supports index creation in materialized views.
- Supports the Analyze operation in materialized views.

Unsupported Scenarios

- Multi-table join plans and subquery plans are not supported in materialized views.
- Except for a few ALTER operations, most DDL operations cannot be performed on base tables in materialized views.
- Materialized views cannot be added, deleted, or modified. They support only query statements.
- The temporary table, hashbucket, unlog, or partitioned table cannot be used to create materialized views.
- Materialized views cannot be created in nested mode (that is, a materialized view cannot be created in another materialized view).
- The column-store tables are not supported. Only row-store tables are supported.
- Materialized views of the UNLOGGED type are not supported, and the WITH syntax is not supported.

Constraints

- If the materialized view definition is UNION ALL, each subquery needs to use a different base table.
- When a fast-refresh materialized view is created, fully refreshed, or deleted, a high-level lock is added to the base table. If the materialized view is defined as UNION ALL, pay attention to the service logic to avoid deadlock.

17 GUC Parameters

17.1 GUC Parameter Usage

A database provides many operation parameters. Configurations of these parameters affect the behavior of the database system. Before modifying these parameters, learn the impact of these parameters on the database. Otherwise, unexpected results may occur.

You are advised to modify some parameters on the GaussDB console. If the parameters cannot be modified on the console, evaluate the risks and contact customer service.

Precautions

- If the value range of a parameter is a string, the string should comply with the naming conventions of the path and file name in the OS running the target database.
- If the maximum value of a parameter is *INT_MAX*, the maximum parameter value varies by OS.
- If the maximum value of a parameter is *DBL_MAX*, the maximum parameter value varies by OS.

17.2 File Location

After a database has been installed, three configuration files (**postgresql.conf**, **pg_hba.conf**, and **pg_ident.conf**) are automatically generated and saved in the data directory. You can use the methods described in this section to change the names and save paths of these configuration files.

When changing the storage directory of a configuration file, set **data_directory** in **postgresql.conf** to the actual data directory.

NOTICE

If a configuration file is incorrectly modified, the database will be seriously affected. Do not modify the configuration files mentioned in this section after installation.

data_directory

Parameter description: Specifies the GaussDB **data** directory. Only users with the **sysadmin** permission can access this parameter. You can set this parameter using one of the following methods:

- Set it when you install the GaussDB.
- This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string, consisting of one or more characters

Default value: Specify this parameter during installation. If this parameter is not specified during installation, the database is not initialized by default.

config_file

Parameter description: Specifies the configuration file (**postgresql.conf**) of the primary server.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string, consisting of one or more characters

Default value: **postgresql.conf** (The absolute directory of this file may be displayed in the actual situation.)

hba_file

Parameter description: Specifies the configuration file (**pg_hba.conf**) for host-based authentication (HBA). This parameter can be specified only in the **postgresql.conf** file and can be accessed only by users with the **sysadmin** permission.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string

Default value: **pg_hba.conf** (The absolute directory of this file may be displayed in the actual situation.)

ident_file

Parameter description: Specifies the name of the configuration file (**pg_ident.conf)** for client authentication. Only users with the **sysadmin** permission can access this parameter.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string

Default value: **pg_ident.conf** (The absolute directory of this file may be displayed in the actual situation.)

external_pid_file

Parameter description: Specifies the extra PID file that can be used by the server management program. Only the sysadmin user can access this parameter.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

This parameter takes effect only after the database restarts.

Value range: a string

Default value: empty

enable_default_cfunc_libpath

Parameter description: Specifies whether the default path is used for the .so file when the C function is created in GaussDB.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

on: indicates that the .so file must be placed in the specified directory (**$libdir/proc_srclib**) when the C function is created.

off: indicates that the .so file can be stored in any accessible directory when the C function is created.

Default value: **on**

NOTICE

If this parameter is set to **off**, the .so file can be placed in any accessible directory or the .so file provided by the system can be used, which poses security risks. Therefore, you are not advised to set this parameter to **off**.

17.3 Connection and Authentication

17.3.1 Connection Settings

This section describes parameters related to client-server connection modes.

listen_addresses

Parameter description: Specifies the TCP/IP address of the client for a server to listen on.

This parameter specifies the IP address used by the GaussDB server for listening, for example, IPv4. Multiple NICs may exist on the host and each NIC can be bound to multiple IP addresses. This parameter specifies the IP addresses to which GaussDB is bound. The client can use the IP address specified by this parameter to connect to or send requests to GaussDB.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range:

- Host name or IP address. Multiple values are separated with commas (,).
- Asterisk (*) or **0.0.0.0**, indicating that all IP addresses will be listened to, which is not recommended due to potential security risks. This parameter must be used together with valid addresses (for example, the local IP address). Otherwise, the build may fail. In primary/standby mode, if the value is set to ***** or **0.0.0.0**, the value of **localport** in the **postgresql.conf** file of the database on the primary node cannot be the value of **dataPortBase + 1**. Otherwise, the database cannot be started.
- If the parameter is not specified, the server does not listen on any IP address. In this case, only Unix domain sockets can be used for database connections.

Default value: After the database instance is installed, the default value is configured according to the IP address of different instances in the **public_cloud.conf** configuration file. The default value for the DN instance is the IP address of the **data.net** NIC.

NOTE

The **public_cloud.conf** file contains the following NIC information: **mgr.net** (management NIC), **data.net** (data NIC), and **virtual.net** (virtual NIC).

local_bind_address

Parameter description: Specifies the host IP address bound to the current node for connecting to other nodes in the database.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Default value: After the database instance is installed, the default value is configured according to the IP address of different instances in the **public_cloud.conf** configuration file. The default value for the DN instance is the IP address of the **data.net** NIC.

NOTE

The **public_cloud.conf** file contains the following NIC information: **mgr.net** (management NIC), **data.net** (data NIC), and **virtual.net** (virtual NIC).

port

Parameter description: Specifies the TCP port listened on by the GaussDB.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

NOTE

This parameter is specified in the configuration file during installation. Do not modify this parameter unless absolutely necessary. Otherwise, database communication will be affected.

Value range: an integer ranging from 1 to 65535

NOTE

- When setting the port number, ensure that the port number is not in use. When setting the port numbers of multiple instances, ensure that the port numbers do not conflict.
- Ports 1 to 1023 are reserved for the operating system. Do not use them.
- When the database instance is installed using the configuration file, pay attention to the ports reserved in the communication matrix in the configuration file. For example, *dataPortBase* + 1 needs to be reserved as the port used by internal tools, and *dataPortBase* + 6 needs to be reserved as the communication port of the streaming engine message queue. (Due to specification changes, the current version no longer supports the current feature. Do not use this feature.) Therefore, during database instance installation, the maximum port number is **65529** for DNs. Ensure that the port number does not conflict with each other.

Default value: **5432** (The actual value is specified in the configuration file during installation.)

max_connections

Parameter description: Specifies the maximum number of concurrent connections to the database. This parameter influences the concurrent processing capability of the database.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer. The minimum value is **10** (greater than **max_wal_senders**). The theoretical maximum value is **262143**. The actual maximum value is a dynamic value, which is calculated using the formula 262143 **– job_queue_processes – autovacuum_max_workers – AUXILIARY_BACKENDS – AV_LAUNCHER_PROCS – max_inner_tool_connections – min(max(newValue/ 4,64),1024)**. The values of **job_queue_processes**, **autovacuum_max_workers** and **max_inner_tool_connections** depend on the settings of the corresponding GUC parameters. **AUXILIARY_BACKENDS** fixes the number of reserved auxiliary threads to **20**. **AV_LAUNCHER_PROCS** fixes the number of reserved launcher threads for autovacuum to **2**.

Default value:

55000 (128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, 96-core CPU/1024 GB memory); **40000** (96-core CPU/768 GB memory); **25000** (64-core CPU/512 GB memory); **24000** (60-core CPU/480 GB memory); **11000** (32-core CPU/256 GB memory); **5000** (16-core CPU/128 GB memory); **2048** (8-core CPU/64 GB memory); **100** (4-core CPU/32 GB memory, 4-core CPU/16 GB memory)

Setting suggestions:

Retain the default value of this parameter on the primary node of the databases.

Impact of incorrect configuration:

- If the value of **max_connections** exceeds the maximum dynamic value, the node fails to be started and the following error message is displayed: "invalid value for parameter "max_connections"". Alternatively, the memory fails to be allocated during the node startup and the following error message is displayed: "Cannot allocate memory".
- If only the value of *max_connections* is increased while the memory parameter is not adjusted in proportion according to the external egress specifications, when the service load is heavy, the memory may be insufficient, and the error message "memory is temporarily unavailable" is displayed.

NOTE

- If the number of connections of the administrator exceeds the value of *max_connections*, the administrator can still connect to the database after the connections are used up by common users. If the number of connections exceeds the value of *sysadmin_reserved_connections*, an error is reported. That is, the maximum number of connections of the administrator is equal to the value of *max_connections* + *sysadmin_reserved_connections*.
- For common users, internal jobs use some connections. Therefore, the value of this parameter is slightly less than that of *max_connections*. The value depends on the number of internal connections.

max_inner_tool_connections

Parameter description: Specifies the maximum number of concurrent connections of a tool which is allowed to connect to the database. This parameter influences the concurrent connection capability of the GaussDB tool.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 1 to *MIN* (which takes the smaller value between **262143** and *max_connections*). For details about how to calculate the value of *max_connections*, see the preceding description.

Default value: **50** for each database node. If the default value is greater than the maximum value supported by the kernel (determined when the **gs_initdb** command is executed), an error message is displayed.

Setting suggestions:

Retain the default value of this parameter on the primary node of the database.

If this parameter is set to a large value, GaussDB requires more System V shared memories or semaphores, which may exceed the default maximum configuration of the OS. In this case, modify the value as needed.

sysadmin_reserved_connections

Parameter description: Specifies the minimum number of connections reserved for administrators. You are advised not to set this parameter to a large value. This parameter is used together with the *max_connections* parameter. The maximum number of connections of the administrator is equal to the value of *max_connections* + *sysadmin_reserved_connections*.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to *MIN* (which takes the smaller value between **262143** and *max_connections*). For details about how to calculate the value of *max_connections*, see the preceding description.

Default value: **3**

Note: When the thread pool function is enabled, if the thread pool is fully occupied, a processing bottleneck occurs. As a result, connections reserved by the administrator cannot be established. In this case, you can use gsql to establish connections through the primary port number + 1 to clear useless sessions.

unix_socket_directory

Parameter description: Specifies the Unix domain socket directory for the GaussDB server to listen to connections from the client.

This parameter is a **POSTMASTER** parameter. Set it based on instructions provided in **Table 10-1**.

The parameter length limit varies by OS. If the length is exceeded, the error "Unix-domain socket path xxx is too long" will be reported.

Value range: a string

Default value: empty. The actual value is specified by the configuration file during installation.

unix_socket_group

Parameter description: Specifies the group of the Unix domain socket (the user of a socket is the user that starts the server). This parameter can work with **unix_socket_permissions** to control socket access.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string. If this parameter is set to an empty string, the default group of the current user is used.

Default value: empty

unix_socket_permissions

Parameter description: Specifies access permissions on the Unix domain socket.

The Unix domain socket uses the usual permission set of the Unix file system. The value of this parameter should be a number (acceptable for the **chmod** and

umask commands). If a user-defined octal format is used, the number must start with 0.

You are advised to set it to **0770** (only allowing access from users connecting to the database and users in the same group as them) or **0700** (only allowing access from users connecting to the database).

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: 0000 to 0777

Default value: **0700**

NOTE

In the Linux OS, a document has one document attribute and nine permission attributes, which consists of the read (r), write (w), and execute (x) permissions of the Owner, Group, and Others groups.

The r, w, and x permissions are represented by the following numbers:

r: 4

w: 2

x: 1

-: 0

The three attributes in a group are accumulative.

For example, **-rwxrwx---** indicates the following permissions:

owner = rwx = 4+2+1 = 7

group = rwx = 4+2+1 = 7

others = --- = 0+0+0 = 0

The permission of the file is 0770.

application_name

Parameter description: Specifies the client name used in the current connection request.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

When a standby node requests to replicate logs on the primary node, if this parameter is not an empty string, it is used as the name of the streaming replication slot of the standby node on the primary node. In this case, if the length of this parameter exceeds 61 bytes, only the first 61 bytes are used as the streaming replication slot name.

Value range: a string

Default value: empty (The actual value is the name of the application connected to the backend.)

connection_info

Parameter description: Specifies the database connection information, including the driver type, driver version, driver deployment path, and process owner.

This parameter is a USERSET parameter used for O&M. You are advised not to change the parameter value.

Value range: a string

Default value: empty

NOTE

- An empty string indicates that the driver connected to the database does not support automatic setting of the **connection_info** parameter or the parameter is not set by users in applications.
- The following is an example of the concatenated value of **connection_info**:
```
{"driver_name":"ODBC","driver_version": "(GaussDB Kernel VxxxRxxxCxx build 290d125f) compiled at 2020-05-08 02:59:43 commit 2143 last mr 131 release","driver_path":"/usr/local/lib/psqlodbcw.so","os_user":"omm"}
```
  driver_name and **driver_version** are displayed by default. Whether **driver_path** and **os_user** are displayed is determined by users. For details, see **Connecting to a Database** and **Configuring a Data Source in the Linux OS**.

17.3.2 Security and Authentication (postgresql.conf)

This section describes parameters about client-to-server authentication.

authentication_timeout

Parameter description: Specifies the longest duration to wait before the client authentication times out. If a client is not authenticated by the server within the period, the server automatically disconnects from the client so that the client does not occupy connection resources.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 1 to 600. The smallest unit is s.

Default value: **1min**

auth_iteration_count

Parameter description: Specifies the number of iterations during the generation of encryption information for authentication.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 2048 to 134217728

Default value: **10000**

NOTICE

If the number of iterations is too small, the password storage security is reduced. If the number of iterations is too large, the performance deteriorates in scenarios involving password encryption, such as authentication and user creation. Set the number of iterations based on actual hardware conditions. You are advised to retain the default value.

session_authorization

Parameter description: Specifies the user ID of the current session.

This parameter is of the USERSET type and can be set only using the **SET SESSION AUTHORIZATION** syntax.

Value range: a string

Default value: **NULL**

session_timeout

Parameter description: Specifies the longest duration allowed when no operations are performed on a client after it is connected to the server.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 86400. The smallest unit is s. **0** indicates that the timeout is disabled.

Default value: 1800s

NOTICE

The gsql client of GaussDB has an automatic reconnection mechanism. For local connection of initialized users, the client reconnects to the server if the connection breaks after the timeout.

ssl

Parameter description: Specifies whether SSL connections are enabled.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that SSL connections are enabled.
- **off** indicates that SSL connections are not enabled.

NOTICE

To enable SSL connections, you also need to configure parameters such as **ssl_cert_file**, **ssl_key_file**, and **ssl_ca_file** and the corresponding files. Incorrect configurations may cause startup failure of the database.

Default value: **on**

require_ssl

Parameter description: Specifies whether the server requires the SSL connection. This parameter is valid only when **ssl** is set to **on**.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the server requires SSL connections.
- **off** indicates that the server does not require SSL connections.

NOTICE

GaussDB supports SSL when a client connects to the primary node of the database. It is recommended that the SSL connection be enabled only on the primary node of the database.

Default value: **off**

ssl_ciphers

Parameter description: Specifies the list of encryption algorithms supported by SSL. Only the sysadmin user can access the list.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string. Separate multiple encryption algorithms by semicolons (;).

Default value: **ALL**

NOTICE

If **ssl_ciphers** is set incorrectly, the database cannot be started properly.

ssl_renegotiation_limit

Parameter description: Specifies the allowed traffic volume over an SSL-encrypted channel before the session key is renegotiated. The renegotiation mechanism reduces the probability that attackers use the password analysis method to crack the key based on a huge amount of data but causes big performance losses. The traffic indicates the sum of transmitted and received traffic. The SSL renegotiation mechanism has been disabled because of potential risks. This parameter is reserved for version compatibility and does not take effect.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 2147483647. The unit is KB. **0** indicates that the renegotiation mechanism is disabled.

Default value: **0**

ssl_cert_file

Parameter description: Specifies the name of the file that contains the SSL server certificate. The relative path is relative to the data directory.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string

Default value: **server.crt**

ssl_key_file

Parameter description: Specifies the name of the file that contains the SSL private key. The relative path is relative to the data directory.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string

Default value: **server.key**

ssl_ca_file

Parameter description: Specifies the name of a file that contains CA information. The relative path is relative to the data directory.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string. If it is an empty string, no CA file is loaded and client certificate verification is not performed.

Default value: **cacert.pem**

ssl_crl_file

Parameter description: Specifies the certificate revocation list (CRL). If the certificate of a client is in the list, the certificate is invalid. A relative path must be used. The path depends on the data directory.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string. An empty string indicates that there is no CRL.

Default value: empty

krb_server_keyfile

Parameter description: Specifies the location of the main configuration file of the Kerberos service. Only the sysadmin user can access the file.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string

Default value: empty

krb_srvname

Parameter description: Specifies the Kerberos service name.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string

Default value: **postgres**

krb_caseins_users

Parameter description: Specifies whether the Kerberos username is case-sensitive.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the Kerberos username is case-insensitive.
- **off** indicates that the Kerberos username is case-sensitive.

Default value: **off**

modify_initial_password

Parameter description: After GaussDB is installed, there is only one initial user account (whose UID is 10) in the database. When a user logs in to the database using this initial user for the first time, this parameter determines whether the password of the initial user needs to be modified.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

If the initial user password is not specified during the installation, the initial user password is empty by default after the installation. Before performing other operations, you need to set the initial user password using the gsql client. This parameter no longer takes effect and is reserved only for compatibility with upgrade scenarios.

Value range: Boolean

- **on** indicates that the password of the initial user needs to be modified upon the first login after the database is successfully installed.
- **off** indicates that the password of the initial user does not need to be modified upon the first login after the database is successfully installed.

Default value: **off**

password_policy

Parameter description: Specifies whether to check the password complexity when you run the **CREATE ROLE/USER** or **ALTER ROLE/USER** command to create or modify an account of GaussDB. For details about the password complexity check policy, see **Setting Password Security Policies**.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

For security purposes, do not disable the password complexity policy.

Value range: **0** and **1**

- **0** indicates that no password complexity policy is enabled.
- **1** indicates that the default password complexity policy is enabled.

Default value: **1**

password_reuse_time

Parameter description: Specifies whether to check the reuse interval of the new password when you run the **ALTER USER** or **ALTER ROLE** command to change a user password. For details about the password reuse policy, see **Setting Password Security Policies**.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

When you change the password, the system checks the values of **password_reuse_time** and **password_reuse_max**.

- If the values of **password_reuse_time** and **password_reuse_max** are both positive numbers, an old password can be reused when it meets either of the reuse restrictions.
- If the value of **password_reuse_time** is **0**, password reuse is restricted based on the number of reuse times, and not on the reuse interval.
- If the value of **password_reuse_max** is **0**, password reuse is restricted based on the reuse interval, and not on the number of reuse times.
- If the values of both **password_reuse_time** and **password_reuse_max** are **0**, password reuse is not restricted.

Value range: a floating point number ranging from 0 to 3650. The unit is day.

- **0** indicates that the password reuse interval is not checked.
- A positive number indicates that a new password cannot be chosen from passwords in history that are newer than the specified number of days.

Default value: **0**

password_reuse_max

Parameter description: Specifies whether to check the reuse times of the new password when you run the **ALTER USER** or **ALTER ROLE** command to change a user password. Only the **sysadmin** user can access this parameter. For details about the password reuse policy, see **Setting Password Security Policies**.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

When you change the password, the system checks the values of **password_reuse_time** and **password_reuse_max**.

- If the values of **password_reuse_time** and **password_reuse_max** are both positive numbers, an old password can be reused when it meets either of the reuse restrictions.
- If the value of **password_reuse_time** is **0**, password reuse is restricted based on the number of reuse times, and not on the reuse interval.
- If the value of **password_reuse_max** is **0**, password reuse is restricted based on the reuse interval, and not on the number of reuse times.
- If the values of both **password_reuse_time** and **password_reuse_max** are **0**, password reuse is not restricted.

Value range: an integer ranging from 0 to 1000

- **0** indicates that the password reuse times are not checked.
- A positive number indicates that the new password cannot be the one whose reuse times exceed the specified number.

Default value: **0**

password_lock_time

Parameter description: Specifies the duration before a locked account is automatically unlocked. For details about the automatic account locking policy, see **Setting Password Security Policies**.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

The locking and unlocking functions take effect only when the values of **password_lock_time** and **failed_login_attempts** are positive numbers.

Value range: a floating point number ranging from 0 to 365. The unit is day.

- **0** indicates that an account is not automatically locked if the password verification fails.
- A positive number indicates the duration after which a locked account is automatically unlocked.

Default value: **1**

failed_login_attempts

Parameter description: Specifies the maximum number of incorrect password attempts before an account is locked. The account will be automatically unlocked after the time specified by **password_lock_time**. Only the sysadmin user can access the account. The automatic account locking policy applies in scenarios such as login and password modification using the **ALTER USER** command. For details about the automatic account locking policy, see **Setting Password Security Policies**.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

The locking and unlocking functions take effect only when the values of **failed_login_attempts** and **password_lock_time** are positive numbers.

Value range: an integer ranging from 0 to 1000

- **0** indicates that the automatic locking function does not take effect.
- A positive number indicates that an account is locked when the number of incorrect password attempts reaches the specified number.

Default value: **10**

password_encryption_type

Parameter description: Specifies the encryption type of a user password. Changing the value of this parameter does not change the password encryption type of existing users. The new encryption type is applied to passwords of new users or passwords modified after the parameter value is changed.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: 0, 1, 2, or 3

- **0** indicates that passwords are encrypted with MD5.
- **1** indicates that passwords are encrypted with SHA-256 and MD5.
- **2** indicates that passwords are encrypted with SHA-256.
- **3** indicates that the passwords are encrypted in sm3 mode.

NOTICE

The MD5 encryption algorithm is not recommended because it has lower security and poses security risks.

Default value: 2

password_min_length

Parameter description: Specifies the minimum length of an account password. Only the sysadmin user can access this parameter.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer. A password can contain 6 to 999 characters.

Default value: **8**

password_max_length

Parameter description: Specifies the maximum length of an account password. Only the sysadmin user can access this parameter.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer. A password can contain 6 to 999 characters.

Default value: **32**

password_min_uppercase

Parameter description: Specifies the minimum number of uppercase letters that an account password must contain. Only the sysadmin user can access this parameter.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 999

- **0** means no limit.
- A positive integer indicates the minimum number of uppercase letters required in a password when you create an account.

Default value: **0**

password_min_lowercase

Parameter description: Specifies the minimum number of lowercase letters that an account password must contain. Only the sysadmin user can access this parameter.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 999

- **0** means no limit.
- A positive integer indicates the minimum number of lowercase letters required in a password when you create an account.

Default value: **0**

password_min_digital

Parameter description: Specifies the minimum number of digits that an account password must contain. Only the sysadmin user can access this parameter.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 999

- **0** means no limit.
- A positive integer indicates the minimum number of digits required in a password when you create an account.

Default value: **0**

password_min_special

Parameter description: Specifies the minimum number of special characters that an account password must contain. Only the sysadmin user can access this parameter.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 999

- **0** means no limit.
- A positive integer indicates the minimum number of special characters required in a password when you create an account.

Default value: **0**

password_effect_time

Parameter description: Specifies the validity period of an account password.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a floating point number ranging from 0 to 999. The unit is day.

- **0** indicates that the validity period restriction is disabled.
- A floating point number from 1 to 999 indicates the number of days for which an account password is valid. When the password is about to expire or has expired, the system prompts the user to change the password.

Default value: **0**

password_notify_time

Parameter description: Specifies how many days in advance a user is notified before a password expires.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 999. The unit is day.

- **0** indicates that the reminder is disabled.
- A positive integer indicates the number of days prior to password expiration that a user will receive a reminder.

Default value: **7**

ssl_cert_notify_time

Parameter description: Specifies the number of days prior to SSL server certificate expiration that a user will receive a reminder. When the SSL certificate is initialized during connection establishment, if the duration from the current time to the certificate expiration time is shorter than the specified value, an expiration notification is recorded in the log.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 7 to 180. The unit is day.

Default value: **90**

17.3.3 Communication Library Parameters

This section describes parameter settings and value ranges for communication libraries.

tcp_keepalives_idle

Parameter description: Specifies the interval for transmitting keepalive signals on an OS that supports the **TCP_KEEPIDLE** socket option. If no keepalive signal is transmitted, the connection is in idle mode.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

- If the OS does not support **TCP_KEEPIDLE**, set this parameter to **0**.
- The parameter is ignored on an OS where connections are established using the Unix domain socket.
- If this parameter is set to **0**, the system value is used.
- This parameter is not shared among different sessions. That is, different session connections may have different values.
- The parameter value in the current session connection, not the value of the GUC copy, is displayed.

Value range: 0 to 3600. The unit is s.

Default value: **60**

tcp_keepalives_interval

Parameter description: Specifies the response time before retransmission on an OS that supports the **TCP_KEEPINTVL** socket option.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: 0 to 180. The unit is s.

Default value:

NOTICE

- If the OS does not support **TCP_KEEPINTVL**, set this parameter to **0**.
- The parameter is ignored on an OS where connections are established using the Unix domain socket.
- If this parameter is set to **0**, the system value is used.
- This parameter is not shared among different sessions. That is, different session connections may have different values.
- The parameter value in the current session connection, not the value of the GUC copy, is displayed.

tcp_keepalives_count

Parameter description: Specifies the number of keepalive signals that can be waited before the GaussDB server is disconnected from the client on an OS that supports the **TCP_KEEPCNT** socket option.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

- If the OS does not support **TCP_KEEPCNT**, set this parameter to **0**.
- The parameter is ignored on an OS where connections are established using the Unix domain socket.
- If this parameter is set to **0**, the system value is used.
- This parameter is not shared among different sessions. That is, different session connections may have different values.
- The parameter value in the current session connection, not the value of the GUC copy, is displayed.

Value range: 0 to 100. **0** indicates that the connection is immediately broken if GaussDB does not receive a keepalived signal from the client.

Default value: 20

tcp_user_timeout

Parameter description: Specifies the maximum duration for which the transmitted data can remain in the unacknowledged state before the TCP connection is forcibly closed when the GaussDB sends data on the OS that supports the TCP_USER_TIMEOUT socket option.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

- If the OS does not support the TCP_USER_TIMEOUT option, the value of this parameter does not take effect. The default value is **0**.
- The parameter is ignored on an OS where connections are established using the Unix domain socket.

Value Range: 0 to 3600000. The unit is ms. The value **0** indicates that the value is set based on the OS.

Default value: **0**

Note that the effective result of this parameter varies according to the OS kernel.

- For AArch64 EulerOS (Linux kernel version: 4.19), the timeout interval is the value of this parameter.
- For x86 EulerOS 2.5 (Linux kernel version: 3.10), the timeout interval is not the set value of this parameter but the maximum value in different ranges. That is, the maximum value of the total Linux TCP retransmission duration range to which the set value of **tcp_user_timeout** belongs. For example, if **tcp_user_timeout** is set to **40000**, the actual total retransmission duration is 51 seconds.

Table 17-1 Value of tcp_user_timeout for x86 EulerOS 2.5 (Linux kernel version: 3.10)

Number of Linux TCP Retransmission Times	Total Linux TCP Retransmission Duration Range (s)	Example of tcp_user_timeout (ms)	Actual Linux TCP Retransmission Duration (s)
1	(0.2,0.6]	400	0.6
2	(0.6,1.4]	1000	1.4
3	(1.4,3]	2000	3
4	(3,6.2]	4000	6.2
5	(6.2,12.6]	10000	12.6
6	(12.6,25.4]	20000	25.4
7	(25.4,51]	40000	51
8	(51,102.2]	80000	102.2
9	(102.2,204.6]	150000	204.6
10	(204.6,324.6]	260000	324.6
11	(324.6,444.6]	400000	444.6

Note: The duration of each TCP retransmission increases exponentially with the number of retransmission times until it reaches 120s. When the duration of a TCP retransmission reaches 120s, the duration of each subsequent retransmission remains 120s.

comm_proxy_attr

Parameter description: Specifies the parameters related to the communication proxy library.

NOTE

- This parameter applies only to the centralized ARM standalone system running EulerOS 2.9.
- This function takes effect when the thread pool is enabled, that is, **enable_thread_pool** is set to **on**.
- When setting this parameter, you need to set the GUC parameter **local_bind_address** to the IP address of the NIC of the **libos_kni**.
- **Parameter template**: comm_proxy_attr = '{enable_libnet:true, enable_dfx:false, numa_num:4, numa_bind:[[30,31],[62,63],[94,95],[126,127]]}'
- Parameters that need to be configured include:
  - **enable_libnet**: whether to enable the user-mode protocol. The options are as follows: **true** and **false**.
  - **enable_dfx**: whether to enable the communication proxy library view. The options are as follows: **true** and **false**.
  - **numa_num**: number of NUMA nodes in the system. 2P and 4P servers are supported. The value can be: **4** or **8**.
  - **numa_bind**: core binding parameter of the agent thread. Each numa has two CPUs. There are a total of **numa_num** groups. The value range is as follows: [0, Number of CPUs – 1].

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string, consisting of one or more characters

Default value: **none**

17.4 Resource Consumption

17.4.1 Memory

This section describes memory parameters.

NOTICE

These parameters, except **local_syscache_threshold**, take effect only after the database restarts.

memorypool_enable

Parameter description: Specifies whether to enable a memory pool.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the memory pool is enabled.
- **off** indicates that the memory pool is disabled.

Default value: **off**

memorypool_size

Parameter description: Specifies the memory pool size.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 128 x 1024 to *INT_MAX*/2. The unit is KB.

Default value: **512MB**

enable_memory_limit

Parameter description: Specifies whether to enable the logical memory management module.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the logical memory management module is enabled.
- **off** indicates that the logical memory management module is disabled.

Default value: **on**

CAUTION

- If the value of **max_process_memory** minus **shared_buffer** minus **cstore_buffers** minus metadata size is less than 2 GB, GaussDB forcibly sets **enable_memory_limit** to **off**. Metadata is the memory used in GaussDB and is related to some concurrent parameters, such as **max_connections**, **thread_pool_attr** and **max_prepared_transactions**.
- If this parameter is set to **off**, the memory used by the database is not limited. When a large number of concurrent or complex queries are performed, too much memory is used, which may cause OS OOM problems.

max_process_memory

Parameter description: Specifies the maximum physical memory of a database node.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 2 x 1024 x 1024 to *INT_MAX*. The unit is KB.

Default value:

900 GB (128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, 96-core CPU/1024 GB memory); **680 GB** (96-core CPU/768 GB memory); **450 GB** (64-core CPU/512 GB memory); **420 GB** (60-core CPU/480 GB memory); **200 GB** (32-core CPU/256 GB memory); **90 GB** (16-core CPU/128 GB memory); **40 GB** (8-core CPU/64 GB memory); **20 GB** (4-core CPU/32 GB memory); **10 GB** (4-core CPU/16 GB memory)

Setting suggestions:

The value on the database node is determined based on the physical memory of the system and the number of master database nodes deployed on a single node. The recommended calculation formula is as follows: (Physical memory – vm.min_free_kbytes) \ x 0.7 /(1 + Number of primary nodes) This parameter is used to prevent node OOM caused by memory usage increase, ensuring system reliability. **vm.min_free_kbytes** indicates the OS memory reserved for the kernel to receive and send data. Its value is at least 5% of the total memory. That is, max_process_memory = Physical memory x 0.665 / (1 + Number of primary nodes)

CAUTION

If this parameter is set to a value greater than the physical memory of the server, the OS OOM problem may occur.

enable_memory_context_control

Parameter description: Specifies whether to enable the function of checking whether the number of memory contexts exceeds the specified limit. This parameter applies only to the DEBUG version.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the function of checking the number of memory contexts is enabled.
- **off** indicates that the function of checking the number of memory contexts is disabled.

Default value: **off**

uncontrolled_memory_context

Parameter description: Specifies which memory texts will not be checked when the **enable_memory_context_control** parameter is set to **on**. This parameter applies only to the DEBUG version.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

During the query, the title meaning string "MemoryContext white list:" is added to the beginning of the parameter value.

Value range: a string

Default value: empty

shared_buffers

Parameter description: Specifies the size of shared memory used by GaussDB. Increasing the value of this parameter causes GaussDB to request more System V shared memory than the default configuration allows.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 16 to 1073741823. The unit is 8 KB.

The value of **shared_buffers** must be an integer multiple of **BLCKSZ**. Currently, **BLCKSZ** is set to **8 KB**. That is, the value of **shared_buffers** must be an integer multiple of 8 KB. The minimum value changes according to **BLCKSZ**.

Default value:

360 GB (128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, 96-core CPU/1024 GB memory); **270 GB** (96-core CPU/768 GB memory); **180 GB** (64-core CPU/512 GB memory); **160 GB** (60-core CPU/480 GB memory); **80 GB** (32-core CPU/256 GB memory); **36 GB** (16-core CPU/128 GB memory); **16 GB** (8-core CPU/64 GB memory)**8 GB** (4-core CPU/32 GB memory); **4 GB** (4-core CPU/16 GB memory)

Setting suggestions:

1. Set **shared_buffers** to a value less than 40% of the memory. Set it to a large value for row-store tables and a small value for column-store tables. For column-store tables: **shared_buffers** = (Memory of a single server/Number of database nodes on the server) x 0.4 x 0.25
2. If **shared_buffers** is set to a larger value, increase the value of **checkpoint_segments** because a longer period of time is required to write a large amount of new or changed data.
3. If the process fails to be restarted after the value of **shared_buffers** is changed, perform one of the following operations based on the error information:
   a. Adjust the **kernel.shmall**, **kernel.shmmax**, and **kernel.shmmin** OS parameters. For details, see "Preparing for Installation > Modifying OS Configuration > Configuring Other OS Parameters" in *Installation Guide*.
   b. Run the **free -g** command to check whether the available memory and swap space of the OS are sufficient. If the memory is insufficient, manually stop other user programs that occupy much memory.
   c. Do not set **shared_buffers** to an excessively large or small value.

segment_buffers

Parameter description: Specifies the memory size of a GaussDB segment-paged metadata page.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 16 to 1073741823. The unit is 8 KB.

The value of **segment_buffers** must be an integer multiple of **BLCKSZ**. Currently, **BLCKSZ** is set to **8KB**. That is, the value of **segment_buffers** must be an integer multiple of 8 KB. The minimum value changes according to the value of **BLCKSZ**.

Default value: **8MB**

Setting suggestions:

segment_buffers is used to cache the content of segment-paged headers, which is key metadata information. To improve performance, it is recommended that the segment headers of common tables be cached in the buffer and not be replaced. You are advised to set this parameter based on the following formula: Number of tables (including indexes and toast tables) x Number of partitions x 3 + 128. This is because each table (partition) has some extra metadata segments. Generally, a table has three segments. At last, **+ 128** is added because segment- and page-based tablespace management requires a certain number of buffers.

If this parameter is set to a small value, it takes a long time to create a segment-paged table for the first time. Therefore, you are advised to set this parameter to the recommended value.

bulk_write_ring_size

Parameter description: Specifies the size of the ring buffer used by the operation when a large amount of data is written (for example, the copy operation).

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 16384 to 2147483647. The unit is KB.

Default value: **2 GB**

Setting suggestions: Increase the value of this parameter on database nodes if a huge amount of data will be imported.

standby_shared_buffers_fraction

Parameter description: Specifies the **shared_buffers** proportion used on the server where a standby instance is deployed.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a double-precision floating-point number ranging from 0.1 to 1.0

Default value: 1

temp_buffers

Parameter description: Specifies the maximum size of local temporary buffers used by a database session.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

temp_buffers can be modified only before the first use of temporary tables within each session. Subsequent attempts to change the value of this parameter will not take effect on that session.

A session allocates temporary buffers based on the value of **temp_buffers**. If a large value is set in a session that does not require many temporary buffers, only the overhead of one buffer descriptor is added. If a buffer is used, additional 8192 bytes will be consumed for it.

Value range: an integer ranging from 100 to 1073741823. The unit is 8 KB.

Default value: **1MB**

max_prepared_transactions

Parameter description: Specifies the maximum number of transactions that can stay in the **prepared** state simultaneously. Increasing the value of this parameter causes GaussDB to request more System V shared memory than the default configuration allows.

When GaussDB is deployed as an HA system, set this parameter on standby servers to a value greater than or equal to that on primary servers. Otherwise, queries will fail on the standby servers.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 262143

Default value: **200**

NOTE

Generally, explicit PREPARE operations are not required for transactions. If explicit PREPARE operations are performed for transactions, increase the value of this parameter to be greater than the number of concurrent services that require PREPARE to prevent preparation failures.

work_mem

Parameter description: Specifies the amount of memory to be used by internal sort operations and hash tables before they write data into temporary disk files. Sort operations are required for **ORDER BY**, **DISTINCT**, and merge joins. Hash tables are used in hash joins, hash-based aggregation, and hash-based processing of **IN** subqueries.

In a complex query, several sort or hash operations may run in parallel; each operation will be allowed to use as much memory as this parameter specifies. If the memory is insufficient, data will be written into temporary files. In addition, several running sessions could be performing such operations concurrently. Therefore, the total memory used may be many times the value of **work_mem**.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 64 to 2147483647. The unit is KB.

Default value:

256 MB (128-core CPU/1024-GB memory, 104-core CPU/1024-GB memory, 96-core CPU/1024-GB memory, and 96-core CPU/768-GB memory); **128 MB** (64-core CPU/512-GB memory, 60-core CPU/480-GB memory, 32-core CPU/256-GB memory, and 16-core CPU/128-GB memory); **64 MB** (8-core CPU/64-GB memory); **32 MB** (4-core CPU/32-GB memory); **16 MB** (4-core CPU/16-GB memory)

NOTICE

Setting suggestions:

If the physical memory specified by **work_mem** is insufficient, additional operator calculation data will be written into temporary tables based on query characteristics and the degree of parallelism. This reduces performance by five to ten times, and prolongs the query response time from seconds to minutes.

- For complex serial queries, each query requires five to ten associated operations. Set **work_mem** using the following formula: **work_mem** = 50% of the memory/10.
- For simple serial queries, each query requires two to five associated operations. Set **work_mem** using the following formula: **work_mem** = 50% of the memory/5.
- For concurrent queries, set **work_mem** using the following formula: **work_mem** = **work_mem** for serial queries/Number of concurrent SQL statements.
- BitmapScan hash tables are also restricted by **work_mem**, but will not be forcibly flushed to disks. In the case of complete lossify, every 1 MB memory occupied by the hash table corresponds to a 16 GB page of BitmapHeapScan (32 GB for Ustore). After the upper limit of **work_mem** is reached, the memory increases linearly with the data access traffic based on this ratio.

query_mem

Parameter description: Specifies the memory used by a query.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: 0 or an integer greater than 32 MB. The default unit is KB.

Default value: **0**

NOTICE

- If the value of **query_mem** is greater than 0, the optimizer adjusts the memory cost estimate to this value when generating an execution plan.
- If the value is set to a negative value or a positive integer less than 32 MB, the default value **0** is used. In this case, the optimizer does not adjust the estimated query memory.

query_max_mem

Parameter description: Specifies the maximum memory that can be used by a query.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: 0 or an integer greater than 32 MB. The default unit is KB.

Default value: **0**

NOTICE

- If the value of **query_max_mem** is greater than 0, an error is reported when the query memory usage exceeds the value.
- If the value is set to a negative value or a positive integer less than 32 MB, the default value **0** is used. In this case, the optimizer does not limit the query memory.

maintenance_work_mem

Parameter description: Specifies the maximum amount of memory to be used by maintenance operations, such as **VACUUM**, **CREATE INDEX**, and **ALTER TABLE ADD FOREIGN KEY**. This parameter may affect the execution efficiency of **VACUUM**, **VACUUM FULL**, **CLUSTER**, and **CREATE INDEX**.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 1024 to *INT_MAX*. The unit is KB.

Default value:

2 GB (128-core CPU/1024-GB memory, 104-core CPU/1024-GB memory, 96-core CPU/1024-GB memory, 96-core CPU/768-GB memory, 64-core CPU/512-GB memory, and 60-core CPU/480-GB memory); **1 GB** (32-core CPU/256-GB memory); **512 MB** (16-core CPU/128-GB memory); **256 MB** (8-core CPU/64-GB memory); **128 MB** (4-core CPU/32-GB memory); **64 MB** (4-core CPU/16-GB memory)

NOTICE

Setting suggestions:

- The value of this parameter must be greater than that of **work_mem** so that database dumps can be more quickly cleared or restored. In a database session, only one maintenance operation can be performed at a time. Maintenance is usually performed when there are not many running sessions.
- When the **Automatic Vacuuming** process is running, up to **autovacuum_max_workers** times this memory may be allocated. In this case, set **maintenance_work_mem** to a value greater than or equal to that of **work_mem**.
- If a large amount of data is to be clustered, increase the value of this parameter in the session.

psort_work_mem

Parameter description: Specifies the memory capacity to be used for partial sorting in a column-store table before writing to temporary disk files. This parameter can be used for inserting tables having a partial cluster key or index, creating a table index, and deleting or updating a table.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

Such operations may be performed in multiple running sessions concurrently. Therefore, the total memory used may be many times the value of **psort_work_mem**.

Value range: an integer ranging from 64 to 2147483647. The unit is KB.

Default value: **512MB**

max_loaded_cudesc

Parameter description: Specifies the number of cudesc cached in each column when a column-store table is scanned. Increasing the value will improve query performance and increase memory usage, particularly when there are many columns in the column-store table.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

If **max_loaded_cudesc** is set to a large value, memory may be insufficient.

Value range: 100 to 1073741823

Default value: 1024

max_stack_depth

Parameter description: Specifies the maximum safe depth of the GaussDB execution stack. The safety margin is required because the stack depth is not checked in every routine in the server, but only in key potentially-recursive routines, such as expression evaluation.

This parameter is a SUSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 100 to *INT_MAX*. The unit is KB.

Default value:

- If the value of **ulimit -s** minus 640 KB is greater than or equal to 2 MB, the default value of this parameter is **2 MB**.
- If the value of **ulimit -s** minus 640 KB is less than 2 MB, the default value of this parameter is the value of **ulimit -s** minus 640 KB.

NOTICE

When setting this parameter, comply with the following principles:

- The database needs to reserve 640 KB stack depth. Therefore, the ideal value of this parameter is the actual stack size limit enforced by the OS kernel (as set by **ulimit -s**) minus 640 KB.
- If the value of this parameter is greater than the value of **ulimit -s** minus 640 KB before the database is started, the database fails to be started. During database running, if the value of this parameter is greater than the value of **ulimit -s** minus 640 KB, this parameter does not take effect.
- If the value of **ulimit -s** minus 640 KB is less than the minimum value of this parameter, the database fails to be started.
- Setting this parameter to a value greater than the actual kernel limit means that a running recursive function may crash an individual backend process.
- Since not all OSs provide this function, you are advised to set a specific value for this parameter.
- The default value is **2 MB**, which is relatively small and does not easily cause system breakdown.

cstore_buffers

Parameter description: Specifies the shared buffer size used in column-store tables.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 16384 to 1073741823. The unit is KB.

Default value: **1 GB**

Setting suggestions:

Column-store tables use the shared buffer specified by **cstore_buffers** instead of that specified by **shared_buffers**. When column-store tables are mainly used, reduce the value of **shared_buffers** and increase that of **cstore_buffers**.

bulk_read_ring_size

Parameter description: Specifies the size of the ring buffer used by the operation when a large amount of data is queried (for example, during large table scanning).

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 256 to 2147483647. The unit is KB.

Default value: **16MB**

enable_early_free

Parameter description: Specifies whether the operator memory can be released in advance.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the operator memory can be released in advance.
- **off** indicates that the operator memory cannot be released in advance.

Default value: **on**

local_syscache_threshold

Parameter description: Specifies the size of system catalog cache in a session. The current feature is a lab feature. Contact Huawei technical support before using it.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

If **enable_global_plancache** is enabled, **local_syscache_threshold** does not take effect when it is set to a value less than 16 MB to ensure that GPC takes effect. The minimum value is 16 MB.

If **enable_global_syscache** and **enable_thread_pool** are enabled, this parameter indicates the total cache size of the current thread and sessions bound to the current thread.

Value range: an integer ranging from 1 x 1024 to 512 x 1024. The unit is KB.

Default value: 16MB

memory_trace_level

Parameter description: Specifies the control level for recording memory allocation information after the dynamic memory usage exceeds 90% of the maximum dynamic memory. This parameter takes effect only when **use_workload_manager** and **enable_memory_limit** are enabled. This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Valid value: enumerated values

- **none**: indicates that memory allocation information is not recorded.
- **level1**: After the dynamic memory usage exceeds 90% of the maximum dynamic memory, the following information is recorded and the recorded memory information is saved in the *$GAUSSLOG*/**mem_log** directory.
  - Global memory overview.
  - Memory usage of the top 20 memory contexts of the instance, session, and thread types.
  - The **totalsize** and **freesize** columns for each memory context.
- **level2**: After the dynamic memory usage exceeds 90% of the maximum dynamic memory, the following information is recorded and the recorded memory information is saved in the *$GAUSSLOG*/**mem_log** directory.
  - Global memory overview.
  - Memory usage of the top 20 memory contexts of the instance, session, and thread types.
  - The **totalsize** and **freesize** columns for each memory context.
  - Detailed information about all memory allocations in each memory context, including the file where the allocated memory is located, line number, and size.

Default value: **level1**

NOTICE

- If this parameter is set to **level2**, the memory allocation details (file, line, and size) of each memory context are recorded, which greatly affects the performance. Therefore, exercise caution when setting this parameter.
- The recorded memory snapshot information can be queried by using the system function **gs_get_history_memory_d....**.
- The recorded memory context is obtained after all memory contexts of the same type with the same name are summarized.

resilience_memory_reject_percent

Parameter description: Specifies the dynamic memory usage percentage for escape from memory overload. This parameter takes effect only when the GUC parameters **use_workload_manager** and **enable_memory_limit** are enabled. This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string, consisting of one or more characters

This parameter consists of **recover_memory_percent** and **overload_memory_percent**. The meanings of the two parts are as follows:

- **recover_memory_percent**: Percentage of the dynamic memory usage when the memory recovers from overload to the maximum dynamic memory. When the dynamic memory usage is less than the maximum dynamic memory multiplied by the value of this parameter, the overload escape function is disabled and new connections are allowed. The value ranges from 0 to 100. The value indicates a percentage.
- **overload_memory_percent**: Percentage of the dynamic memory usage to the maximum dynamic memory when the memory is overloaded. When the dynamic memory usage is greater than the maximum dynamic memory multiplied by the value of this parameter, the current memory is overloaded. In this case, the overload escape function is triggered to kill sessions and new connections are prohibited. The value ranges from 0 to 100. The value indicates a percentage.

Default value: **'0,0'**, indicating that the escape from memory overload function is disabled.

Example:

```
resilience_memory_reject_percent = '70,90'
```

When the memory usage exceeds 90% of the upper limit, new connections are forbidden and stacked sessions are killed. When the memory usage is less than 70% of the upper limit, session killing is stopped and new connections are allowed.

NOTICE

- You can query the maximum dynamic memory and used dynamic memory in the **gs_total_memory_detail** view. **max_dynamic_memory** indicates the maximum dynamic memory, and **dynamic_used_memory** indicates the used dynamic memory.
- If this parameter is set to a small value, the escape from memory overload process is frequently triggered. As a result, ongoing sessions are forcibly logged out, and new connections fail to be connected for a short period of time. Therefore, exercise caution when setting this parameter based on the actual memory usage.
- The values of **recover_memory_percent** and **overload_memory_percent** can be **0** at the same time. In addition, the value of **recover_memory_percent** must be smaller than that of **overload_memory_percent**. Otherwise, the setting does not take effect.

17.4.2 Disk Space

This section describes the disk space parameters, which are used to set limits on the disk space for storing temporary files.

sql_use_spacelimit

Parameter description: Specifies the space size for files to be flushed to disks when a single SQL statement is executed on a single database node. The managed space includes the space occupied by ordinary tables, temporary tables, and intermediate result sets to be flushed to disks.

This parameter is a **USERSET** parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from –1 to 2147483647. The unit is KB. **–1** indicates no limit.

Default value: –1

temp_file_limit

Parameter description: Specifies the limit on the size of a temporary file spilled to disk in a session. The temporary file can be a sort or hash temporary file, or the storage file for a held cursor.

This is a session-level setting.

This parameter is a SUSET parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

This parameter does not apply to disk space used for temporary tables during the SQL query process.

Value range: an integer ranging from –1 to 2147483647. The unit is KB. **–1** indicates no limit.

Default value: **–1**

17.4.3 Kernel Resource Usage

This section describes kernel resource parameters. Whether these parameters take effect depends on OS settings.

max_files_per_process

Parameter description: Specifies the maximum number of simultaneously open files allowed by each server process. If the kernel is enforcing a proper limit, setting this parameter is not required.

However, on some platforms, such as most Berkeley Software Distribution (BSD) systems, the kernel allows individual processes to open many more files than the system can support. If the message "Too many open files" is displayed, set this parameter to a smaller value. Generally, the system must meet this requirement: Number of file descriptors >= Maximum number of concurrent statements x Number of database nodes x **max_files_per_process** x 3

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 25 to 2147483647

Default value: 1024

shared_preload_libraries

Parameter description: Specifies one or more shared libraries to be preloaded at server start. If multiple libraries are to be loaded, separate their names using commas (,). Only the sysadmin user can access this parameter. For example, **$libdir/mylib** will cause **mylib.so** (or on some platforms, **mylib.sl**) to be preloaded before the loading of the standard library directory.

You can preinstall the GaussDB's stored procedure library using the **$libdir/pl***XXX* syntax as described in the preceding text. *XXX* can only be **pgsql**, **perl**, **tcl**, or **python**.

By preloading a shared library and initializing it as required, the library startup time is avoided when the library is first used. However, the time to start each new server process may increase, even if that process never uses the library. Therefore, set this parameter only for libraries that will be used in most sessions.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

- If a specified library is not found, the GaussDB service will fail to start.
- Each GaussDB-supported library has a special mark that is checked to guarantee compatibility. Therefore, libraries that do not support GaussDB cannot be loaded in this way.

Value range: a string

Default value: **security_plugin**

17.4.4 Cost-based Vacuum Delay

This feature allows administrators to reduce the I/O impact of the **VACUUM** and **ANALYZE** statements on concurrent database activities. It is often more important to prevent maintenance statements, such as **VACUUM** and **ANALYZE**, from affecting other database operations than to run them quickly. Cost-based vacuum delay provides a way for administrators to achieve this purpose.

NOTICE

Certain vacuum operations hold critical locks and should be complete as quickly as possible. In GaussDB, cost-based vacuum delays do not take effect during such operations. To avoid uselessly long delays in such cases, the actual delay is the larger of the two calculated values:

- **vacuum_cost_delay** x **accumulated_balance/vacuum_cost_limit**
- **vacuum_cost_delay** x 4

Background

During the execution of the **ANALYZE | ANALYSE** and **VACUUM** statements, the system maintains an internal counter that keeps track of the estimated cost of the various I/O operations that are performed. When the accumulated cost reaches a limit (specified by **vacuum_cost_limit**), the process performing the operation will sleep for a short period of time (specified by **vacuum_cost_delay**). Then, the counter resets and the operation continues.

By default, this feature is disabled. To enable this feature, set **vacuum_cost_delay** to a non-zero value.

vacuum_cost_delay

Parameter description: Specifies the length of time that a process will sleep when **vacuum_cost_limit** has been exceeded.

On many systems, the effective resolution of the sleep length is 10 milliseconds. Therefore, setting this parameter to a value that is not a multiple of 10 has the same effect as setting it to the next higher multiple of 10.

This parameter is usually set to a small value, such as 10 or 20 milliseconds. Adjusting vacuum's resource consumption is best done by changing other vacuum cost parameters.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 100. A positive number enables cost-based vacuum delay and **0** disables cost-based vacuum delay.

Default value: 1

vacuum_cost_page_hit

Parameter description: Specifies the estimated cost for vacuuming a buffer found in the shared buffer. It represents the cost to lock the buffer pool, look up the shared hash table, and scan the content of the page.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 10000

Default value: **1**

vacuum_cost_page_miss

Parameter description: Specifies the estimated cost for vacuuming a buffer read from the disk. It represents the cost to lock the buffer pool, look up the shared hash table, read the desired block from the disk, and scan the block.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 10000

Default value: 10

vacuum_cost_page_dirty

Parameter description: Specifies the estimated cost charged when vacuum modifies a block that was previously clean. It represents the extra cost required to update the dirty block out to the disk again.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 10000

Default value: 20

vacuum_cost_limit

Parameter description: Specifies the cost limit. The vacuuming process will sleep if this limit is exceeded.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 1 to 10000

Default value: 1000

17.4.5 Background Writer

This section describes background writer parameters. The background writer process is used to write dirty data (new or modified data) in shared buffers to disks. This mechanism ensures that database processes seldom or never need to wait for a write action to occur when handling user queries.

It also mitigates performance deterioration caused by checkpoints because only a few of dirty pages need to be flushed to the disk when the checkpoints arrive. This mechanism, however, increases the overall net I/O load because while a repeatedly-dirtied page may otherwise be written only once per checkpoint interval, the background writer may write it several times as it is dirtied in the same interval. In most cases, continuous light loads are preferred, instead of periodical load peaks. The parameters discussed in this section can be set based on actual requirements.

bgwriter_delay

Parameter description: Specifies the interval at which the background writer writes dirty shared buffers. Each time, the backend write process initiates write operations for some dirty buffers. In full checkpoint mode, the **bgwriter_lru_maxpages** parameter is used to control the amount of data to be written each time, and the process is restarted after *bgwriter_delay* ms hibernation. In incremental checkpoint mode, the number of target idle buffer pages is calculated based on the value of **candidate_buf_percent_target**. If the number of idle buffer pages is insufficient, a batch of pages is flushed to disks every *bgwriter_delay* ms. The number of flushed pages is calculated based on the target difference percentage. The maximum number of flushed pages is limited by **max_io_capacity**.

In many systems, the effective resolution of sleep delays is 10 milliseconds. Therefore, setting this parameter to a value that is not a multiple of 10 has the same effect as setting it to the next higher multiple of 10.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 10 to 10000. The unit is millisecond.

Default value: **2s**

Setting suggestion: Reduce this value in slow data writing scenarios to reduce the checkpoint load.

candidate_buf_percent_target

Parameter description: Specifies the expected percentage of available buffers in the shared_buffer memory buffer in the candidate buffer chain when the incremental checkpoint is enabled. If the number of available buffers in the current candidate chain is less than the target value, the bgwriter thread starts flushing dirty pages that meet the requirements.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a double-precision floating point number ranging from 0.1 to 0.85

Default value: **0.3**

bgwriter_lru_maxpages

Parameter description: Specifies the number of dirty buffers the background writer can write in each round.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 1000

NOTE

When this parameter is set to **0**, the background writer is disabled. This setting does not affect checkpoints.

Default value: **100**

bgwriter_lru_multiplier

Parameter description: Specifies the coefficient used to estimate the number of dirty buffers the background writer can write in the next round.

The number of dirty buffers written in each round depends on the number of buffers used by server processes during recent rounds. The estimated number of buffers required in the next round is calculated using the following formula: Average number of recently used buffers x **bgwriter_lru_multiplier**. The background writer writes dirty buffers until sufficient, clean and reusable buffers are available. The number of buffers the background writer writes in each round is always equal to or less than the value of **bgwriter_lru_maxpages**.

Therefore, the value **1.0** of **bgwriter_lru_multiplier** represents a just-in-time policy of writing exactly the number of dirty buffers predicted to be required. Larger values provide some cushion against spikes in demand, whereas smaller values intentionally leave more writes to be done by server processes.

Smaller values of **bgwriter_lru_maxpages** and **bgwriter_lru_multiplier** reduce the extra I/O load caused by the background writer, but make it more likely that server processes will have to issue writes for themselves, delaying interactive queries.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a floating point number ranging from 0 to 10

Default value: 2

pagewriter_thread_num

Parameter description: Specifies the number of threads for background page flushing after the incremental checkpoint is enabled. Dirty pages are flushed in sequence to disks, promoting recovery points.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 1 to 16

Default value: **4**

dirty_page_percent_max

Parameter description: Specifies the percentage of dirty pages to **shared_buffers** after the incremental checkpoint is enabled. When the value of this parameter is reached, the background page flush thread flushes dirty pages based on the maximum value of max_io_capacity.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a floating point number ranging from 0.1 to 1

Default value: **0.9**

pagewriter_sleep

Parameter description: Specifies the interval for the pagewriter thread to flush dirty pages to disks after the incremental checkpoint is enabled. When the ratio of dirty pages to shared_buffers reaches dirty_page_percent_max, the number of pages in each batch is calculated based on the value of max_io_capacity. In other cases, the number of pages in each batch decreases proportionally.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 3600000. The unit is ms.

Default value: **2000ms(2s)**

max_io_capacity

Parameter description: Specifies the maximum I/O per second for the backend write process to flush pages in batches. Set this parameter based on the service scenario and disk I/O capability of the host. If the RTO is short or the data volume is much larger than the shared memory, and the service access data volume is random, the value of this parameter cannot be too small. A small parameter value reduces the number of pages flushed by the backend write process. If a large number of pages are eliminated due to service triggering, the services are affected.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 30720 to 10485760. The unit is KB.

Default value: **512000 KB** (500 MB)

enable_consider_usecount

Parameter description: Specifies whether the backend thread considers the page popularity during page replacement. You are advised to enable this parameter in large-capacity scenarios.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on**/**true**: The page popularity is considered.
- **off**/**false**: The page popularity is not considered.

Default value: **off**

dw_file_num

Parameter description: Specifies the number of doublewrite files to be written in batches. The value is related to **pagewriter_thread_num** and cannot be greater than **pagwriter_thread_num**. If the value is too large, it will be corrected to the value of **pagewriter_thread_num**.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 1 to 16

Default value: **1**

dw_file_size

Parameter description: Specifies the size of each doulewrite file.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer, in the range [32,256]

Default value: **256**

17.4.6 Asynchronous I/O Operations

enable_adio_debug

Parameter description: Specifies whether OM personnel are allowed to generate some ADIO logs to locate ADIO issues. This parameter is used only by developers. Common users are advised not to use it. This function is no longer supported in the current version due to specification changes. Do not use this function.

This parameter is a SUSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** or **true** indicates that generation of logs is allowed.
- **off** or **false** indicates that generation of logs is not allowed.

Default value: **off**

NOTE

This parameter cannot be enabled on in the current version. Even if it is manually enabled, the system automatically disables it.

enable_adio_function

Parameter description: Specifies whether to enable the ADIO function. This function is no longer supported in the current version due to specification changes. Do not use this function.

NOTE

The current version does not support the asynchronous I/O function. This function is disabled by default. Do not modify the setting.

Value range: Boolean

- **on** or **true** indicates that the function is enabled.
- **off** or **false** indicates that the function is disabled.

Default value: **off**

enable_fast_allocate

Parameter description: Specifies whether the quick disk space allocation is enabled. This function is no longer supported in the current version due to specification changes. Do not use this function.

This parameter is a SUSET parameter. Set it based on instructions provided in **Table 10-1**. This function can be enabled only in the XFS file system.

Value range: Boolean

- **on** or **true** indicates that the function is enabled.
- **off** or **false** indicates that the function is disabled.

Default value: **off**

prefetch_quantity

Parameter description: Specifies the amount of the I/O that the row-store prefetches using the ADIO. This function is no longer supported in the current version due to specification changes. Do not use this function.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 128 to 131072. The unit is 8 KB.

Default value: **32 MB** (4096 x 8 KB)

backwrite_quantity

Parameter description: Specifies the amount of I/O that the row-store writes using the ADIO. This function is no longer supported in the current version due to specification changes. Do not use this function.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 128 to 131072. The unit is 8 KB.

Default value: **8 MB** (1024 x 8 KB)

cstore_prefetch_quantity

Parameter description: Specifies the amount of I/O that the column-store prefetches using the ADIO. This function is no longer supported in the current version due to specification changes. Do not use this function.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 1024 to 1048576. The unit is KB.

Default value: **32 MB**

cstore_backwrite_quantity

Parameter description: Specifies the amount of I/O that the column-store writes using the ADIO. This function is no longer supported in the current version due to specification changes. Do not use this function.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 1024 to 1048576. The unit is KB.

Default value: **8MB**

cstore_backwrite_max_threshold

Parameter description: Specifies the maximum amount of buffer I/O that the column-store writes in the database using the ADIO. This function is no longer supported in the current version due to specification changes. Do not use this function.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 4096 to *INT_MAX*/2. The unit is KB.

Default value: **2 GB**

fast_extend_file_size

Parameter description: Specifies the disk size that the row-store pre-scales using the ADIO. This function is no longer supported in the current version due to specification changes. Do not use this function.

This parameter is a SUSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 1024 to 1048576. The unit is KB.

Default value: **8MB**

effective_io_concurrency

Parameter description: Specifies the number of requests that can be simultaneously processed by a disk subsystem. For the RAID array, the parameter value must be the number of disk drive spindles in the array. This function is no longer supported in the current version due to specification changes. Do not use this function.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 1000

Default value: **1**

checkpoint_flush_after

Parameter description: Specifies the threshold for the number of pages flushed by the checkpointer thread. If the threshold is exceeded, the operating system is instructed to flush the pages cached in the operating system asynchronously. In GaussDB, the disk page size is 8 KB.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 256. **0** indicates that the asynchronous flush function is disabled. For example, if the value is **32**, the checkpointer thread continuously writes 32 disk pages (that is, 32 x 8 = 256 KB) before asynchronous flush.

Default value: **256 KB**

bgwriter_flush_after

Parameter description: Specifies the threshold for the number of pages flushed by the background writer thread. If the threshold is exceeded, the background writer thread instructs the operating system to asynchronously flush the pages cached in the operating system to disks. In GaussDB, the disk page size is 8 KB.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 256. **0** indicates that the asynchronous flush function is disabled. The size of a single page is 8 KB. For example, if the value is **64**, the background writer thread continuously writes 64 disk pages (that is, 64 x 8 = 512 KB) before asynchronous flush.

Default value: **512 KB** (64 pages)

backend_flush_after

Parameter description: Specifies the threshold for the number of pages flushed by the background writer thread. If the number of pages exceeds the threshold, the background writer thread instructs the operating system to asynchronously flush the pages cached in the operating system to disks. In GaussDB, the disk page size is 8 KB.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 256. **0** indicates that the asynchronous flush function is disabled. The size of a single page is 8 KB. For example, if the value is **64**, the backend thread continuously writes 64 disk pages (that is, 64 x 8 = 512 KB) before asynchronous flush.

Default value: **0**

17.5 Write Ahead Log

17.5.1 Settings

wal_level

Parameter description: Specifies the level of information to be written to the WAL. The value cannot be empty or commented out.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

- To enable WAL archiving and data streaming replication between primary and standby servers, set this parameter to **archive**, **hot_standby**, or **logical**.
- This parameter can be set to **minimal** only in a single-server system. When the parameter value is **minimal**, **archive_mode** must be set to **off**, **hot_standby** must be set to **off**, and **max_wal_senders** must be set to **0**. Otherwise, the database startup fails.
- If this parameter is set to **archive**, **hot_standby** must be set to **off**. Otherwise, the database startup fails. However, **hot_standby** cannot be set to **off** in a primary-standby deployment. For details, see the description of the **hot_standby** parameter.

Value range: enumerated values

- **minimal**

  Advantages: Certain bulk operations (including creating tables and indexes, executing cluster operations, and copying tables) are safely skipped in logging, which can make those operations much faster.

  Disadvantages: WALs contain only basic information required for recovery from a database server crash or an emergency shutdown. Data cannot be restored from archived WALs.

- **archive**

  Adds logging required for WAL archiving, supporting the database restoration from archives.

- **hot_standby**
  - Further adds information required to run SQL queries on a standby server and takes effect after a server restart.
  - To enable read-only queries on a standby server, the **wal_level** parameter must be set to **hot_standby** on the primary server and the same value must be set on the standby server. There are few measurable differences in performance between using **hot_standby** and **archive** levels. However, feedback is welcome if any differences in their impacts on product performance are noticeable.

- **logical**

  This parameter indicates that WALs support logical replication.

Default value: **hot_standby**

fsync

Parameter description: Specifies whether the GaussDB server uses the **fsync()** function (see **wal_sync_method**) to ensure that updates can be written to disks in a timely manner.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

- Using the **fsync()** function ensures that the data can be recovered to a known state when an OS or a hardware crashes.
- Setting this parameter to **off** may result in unrecoverable data corruption in a system crash.

Value range: Boolean

- **on** indicates that the **fsync()** function is used.
- **off** indicates that the **fsync()** function is not used.

Default value: **on**

synchronous_commit

Parameter description: Specifies the synchronization mode of the current transaction.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Generally, logs generated by a transaction are synchronized in the following sequence:

1. The primary node writes the log content to the local memory.
2. The primary node writes the logs in the local memory to the local file system.
3. The primary node flushes logs in the local file system to disks.
4. The primary node sends the log content to the standby node.
5. The standby node receives the log content and saves it to the local memory.
6. The standby node writes logs in the local memory to the local file system.
7. The standby node flushes logs in the local file system to disks.
8. The standby node replays logs to complete the incremental update of data files.

Value range: enumerated values

- **on**: The primary node waits for the standby node to flush logs to disks before committing a transaction.
- **off**: The primary node commits transactions without waiting for the primary node to flush logs to disks. This mode is also called asynchronous commit.
- **local**: The primary node waits for the primary node to flush logs to disks before committing a transaction. This mode is also called local commit.
- **remote_write**: The primary node waits for the standby node to write logs to the file system before committing a transaction. (The logs do not need to be flushed to disks.)
- **remote_receive**: The primary node waits for the standby node to receive logs before committing a transaction. (The logs do not need to be written to the file system.)
- **remote_apply**: The primary node waits for the standby node to complete log replay before committing a transaction.
- **true**: same as **on**.
- **false**: same as **off**.
- **yes**: same as **on**.
- **no**: same as **off**.
- **1**: same as **on**.
- **0**: same as **off**.
- **2**: same as **remote_apply**.

Default value: **on**

wal_sync_method

Parameter description: Specifies the method used for forcing WAL updates out to disk.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

If **fsync** is set to **off**, the setting of this parameter does not take effect because WAL file updates will not be forced out to disk.

Value range: enumerated values

- **open_datasync** indicates that WAL files are opened with the **O_DSYNC** option.
- fdatasync indicates that **fdatasync()** is called at each commit. (SUSE 10 and SUSE 11 are supported.)
- **fsync_writethrough** indicates that **fsync()** is called at each commit to force data in the buffer to be written to the disk.

  NOTE

  wal_sync_method can be set to **fsync_writethrough** on a Windows platform, but this setting has the same effect as setting the parameter to **fsync**.
- **fsync** indicates that **fsync()** is invoked at each commit (SUSE 10 and SUSE 11 are supported).
- **open_sync** indicates that the **open()** with the O_SYNC option is used to write WAL files (SUSE 10 and SUSE 11 are supported).

  NOTE

  Not all platforms support the preceding parameters.

Default value: fdatasync

full_page_writes

Parameter description: Specifies whether the GaussDB server writes the entire content of each disk page to WALs during the first modification of that page after a checkpoint. When the incremental checkpoint function and **enable_double_write** are enabled at the same time, **full_page_writes** is not used.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

- This parameter is needed because a page write that is in process during an OS crash might be only partially completed, leading to an on-disk page that contains a mix of old and new data. The row-level change data normally stored in WALs will not be enough to completely restore such a page during post-crash recovery. Storing the full page image guarantees that the page can be correctly restored, but at the price of increasing the amount of data that must be written to WALs.
- Setting this parameter to **off** might lead to unrecoverable data corruption after a system failure. It might be safe to set this parameter to **off** if you have hardware (such as a battery-backed disk controller) or file-system software (such as ReiserFS 4) that reduces the risk of partial page writes to an acceptably low level.

Value range: Boolean

- **on** indicates that this feature is enabled.
- **off** indicates that this feature is disabled.

Default value: **on**

wal_log_hints

Parameter description: Specifies whether to write an entire page to WALs during the first modification of that page after a checkpoint, even for non-critical modifications of so-called hint bits. You are advised not to modify the setting.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the entire page is written to WALs.
- **off** indicates that the entire page is not written to WALs.

Default value: **on**

wal_buffers

Parameter description: Specifies the number of **XLOG_BLCKSZ** used for storing WAL data. The size of each **XLOG_BLCKSZ** is 8 KB.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: –1 to 2^{18}. The minimum value is –1 and the maximum value is 262144. The unit is 8 KB.

- If this parameter is set to –1, the value of **wal_buffers** is automatically adjusted to 1/64 of **shared_buffers**. The minimum value is 8 **XLOG_BLCKSZ** and the maximum value is 2048 **XLOG_BLCKSZ**, if the automatically adjusted value is less than the minimum value, the value is adjusted to the minimum value. If the automatically adjusted value is greater than the maximum value, the value is adjusted to the maximum value.

- If this parameter is set to a value other than **–1** and smaller than **4**, the value **4** is used.

Default value:

1 GB (128-core CPU/1024-GB memory, 104-core CPU/1024-GB memory, 96-core CPU/1024-GB memory, 96-core CPU/768-GB memory, 64-core CPU/512-GB memory, 60-core CPU/480-GB memory, and 32-core CPU/256-GB memory); **512 MB** (16-core CPU/128-GB memory); **256 MB** (8-core CPU/64-GB memory); **128 MB** (4-core CPU/32-GB memory); **64 MB** (4-core CPU/16-GB memory)

Setting suggestions: The content of the WAL buffers is written to disks at every transaction commit, so extremely large values are unlikely to provide a significant increase in system performance. However, setting this parameter to hundreds of megabytes can improve the disk write performance on a server to which a large number of transactions are committed at the same time. The default value meets user requirements in most cases.

wal_writer_delay

Parameter description: Specifies the delay between activity rounds for the WAL writer.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

A longer delay might lead to insufficient WAL buffer and a shorter delay leads to continuously writing of the WALs, thereby increasing the load of disk I/O.

Value range: an integer ranging from 1 to 10000. The unit is millisecond.

Default value: **200 ms**

commit_delay

Parameter description: Specifies the duration for committed data to be stored in the WAL buffer.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

- When this parameter is set to a non-zero value, the committed transaction is stored in the WAL buffer instead of being written to the WAL immediately. Then the WAL writer process flushes the buffer out to disks periodically.
- If system load is high, other transactions are probably ready to be committed within the delay. If no other transactions are ready to be committed, the delay is a waste of time.

Value range: an integer ranging from 0 to 100000. The unit is µs. **0** indicates no delay.

Default value: **0**

commit_siblings

Parameter description: Specifies a threshold on the number of concurrent open transactions. If the number of concurrent open transactions is greater than the value of this parameter, a transaction that initiates a commit request will wait for a period of time specified by **commit_delay**. Otherwise, this transaction is written into WALs immediately.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 1000

Default value: **5**

wal_block_size

Parameter description: Specifies the size of a WAL disk block.

This parameter is a fixed INTERNAL parameter and cannot be modified.

Value range: an integer. The unit is byte.

Default value: **8192**

wal_segment_size

Parameter description: Specifies the size of a WAL segment file.

This parameter is a fixed INTERNAL parameter and cannot be modified.

Value range: an integer. The unit is 8 KB.

Default value: **16 MB** (2048 x 8 KB)

walwriter_cpu_bind

Parameter description: Specifies the CPU core bound to the WAL write thread. This parameter is used together with the thread pool parameter.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from –1 to the number of cores minus 1

Default value: **–1**

walwriter_sleep_threshold

Parameter description: Specifies the number of times that idle Xlogs are refreshed before the xlogflusher enters the sleep state. If the number of times reaches the threshold, the xlogflusher enters the sleep state.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 1 to 50000

Default value: **500** (128-core CPU/1024-GB memory, 104-core CPU/1024-GB memory, 96-core CPU/1024-GB memory, 96-core CPU/768-GB memory, 64-core CPU/512-GB memory, and 60-core CPU/480-GB memory); **50** (32-core CPU/256-GB memory, 16-core CPU/128-GB memory, 8-core CPU/64-GB memory, 4-core CPU/32-GB memory, and 4-core CPU/16-GB memory)

wal_file_init_num

Parameter description: Specifies the number of Xlog segment files to be created by the WAL writer.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 1000000

Default value: **10** (128-core CPU/1024-GB memory, 104-core CPU/1024-GB memory, 96-core CPU/1024-GB memory, 96-core CPU/768-GB memory, 64-core CPU/512-GB memory, and 60-core CPU/480-GB memory); **0** (32-core CPU/256-GB memory, 16-core CPU/128-GB memory, 8-core CPU/64-GB memory, 4-core CPU/32-GB memory, and 4-core CPU/16-GB memory)

xlog_file_path

Parameter description: Specifies the path of the Xlog shared disk in dual-database instance shared storage scenarios. This parameter is configured by the OM during database system initialization. You are not advised to modify the configuration.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string

Default value: **NULL**

xlog_file_size

Parameter description: Specifies the size of the Xlog shared disk in dual-database instance shared storage scenarios. This parameter is configured by the OM during database system initialization. You are not advised to modify the configuration.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a long integer ranging from 5053733504 to 576460752303423487. The unit is byte.

Default value: **549755813888**

xlog_lock_file_path

Parameter description: Specifies the path of the lock file preempted by the Xlog shared disk in dual-database instance shared storage scenarios. This parameter is configured by the OM during database system initialization. You are not advised to modify the configuration.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string

Default value: **NULL**

force_promote

Parameter description: Specifies whether to enable the forcible switchover function on the standby node.

When a database instance is faulty, the forcible switchover enables the database instance to recover services as soon as possible at the cost of losing some data. This is an escape method used when the database instance is unavailable. You are not advised to trigger this method frequently. You are not advised not to use this function if you are not clear about the impact of data loss on services.

To use this function, you need to enable it on the DN and CM Server and restart the database instance for the setting to take effect. For details about how to enable the forcible switchover function on the standby node, see "Emergency Handling > Performing a Forcible Primary/Standby Switchover" in *Troubleshooting*.

Value range: **0** or **1**

The value **0** indicates that the function is disabled, and the value **1** indicates that the function is enabled.

Default value: **0**

wal_debug

Parameter description: Specifies whether to output WAL-related debugging information. This parameter is available only when **WAL_DEBUG** is enabled during compilation.

This parameter is a SUSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

Default value: **false**

wal_flush_timeout

Parameter description: Specifies the timeout interval for traversing **WalInsertStatusEntryTbl**. It is the maximum wait time for the adaptive Xlog disk flushing I/O to traverse **WalInsertStatusEntryTbl**.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

If the timeout interval is too long, the Xlog flushing frequency may decrease, reducing the Xlog processing performance.

Value range: an integer ranging from 0 to 90000000 (µs)

Default value:**2**

wal_flush_delay

Parameter description: Specifies the wait interval when an entry in the **WAL_NOT_COPIED** state is encountered during **WalInsertStatusEntryTbl** traversal.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 90000000 (µs)

Default value:**1**

17.5.2 Checkpoints

checkpoint_segments

Parameter description: Specifies the minimum number of WAL segment files in the period specified by **checkpoint_timeout**. The size of each log file is 16 MB.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer. The minimum value is **1**.

Increasing the value of this parameter speeds up the export of a large amount of data. Set this parameter based on **checkpoint_timeout** and **shared_buffers**. This parameter affects the number of WAL segment files that can be reused. Generally, the maximum number of reused files in the **pg_xlog** folder is twice the number of **checkpoint_segments**. The reused files are not deleted and are renamed to the WAL segment files which will be later used.

Default value: **1024**

checkpoint_timeout

Parameter description: Specifies the maximum time between automatic WAL checkpoints.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 30 to 3600. The unit is s.

If the value of **checkpoint_segments** is increased, you need to increase the value of this parameter. The increase of these two parameters further requires the increase of **shared_buffers**. Consider all these parameters during setting.

Default value: **15min**

checkpoint_completion_target

Parameter description: Specifies the completion target of each checkpoint, as a fraction of total time between checkpoints.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a double-precision floating point number ranging from 0.0 to 1.0

Default value: **0.5**

NOTE

0.5 indicates that each checkpoint should be complete within 50% of the interval between checkpoints.

checkpoint_warning

Parameter description: Specifies a time in seconds. If the checkpoint interval is close to this time due to filling of checkpoint segment files, a message is sent to the server log to suggest an increase in the **checkpoint_segments** value.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to *INT_MAX*. The unit is second. **0** indicates that the warning is disabled.

Default value: **5min**

Recommended value: **5min**

checkpoint_wait_timeout

Parameter description: Sets the longest time that the checkpoint waits for the checkpointer thread to start.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 2 to 3600. The unit is s.

Default value: **1min**

enable_incremental_checkpoint

Parameter description: Specifies whether to enable incremental checkpointing.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

Default value: **on**

enable_double_write

Parameter description: Specifies whether to enable double writing. When the incremental checkpoint function is enabled and **enable_double_write** is enabled, the **enable_double_write** dual-write feature is used for protection, and **full_page_writes** is not used to prevent half-page write.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

Default value: **on**

incremental_checkpoint_timeout

Parameter description: Specifies the maximum interval between automatic WAL checkpoints when the incremental checkpointing is enabled.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 1 to 3600. The unit is s.

Default value: **1min**

enable_xlog_prune

Parameter description: Specifies whether the primary node reclaims logs if the size of Xlogs exceeds the value of **max_size_for_xlog_prune** when any standby node is disconnected.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- If this parameter is set to **on**, the primary node reclaims logs when any standby node is disconnected.
- If this parameter is set to **off**, the primary node does not reclaim logs when any standby node is disconnected.

Default value: **on**

max_size_for_xlog_prune

Parameter description: This parameter takes effect when **enable_xlog_prune** is enabled. The working mechanism is as follows:

1. If all standby nodes specified by the **replconninfo** series GUC parameters are connected to the primary node, this parameter does not take effect.
2. If any standby node specified by the **replconninfo** series GUC parameters is not connected to the primary node, this parameter takes effect. When the number of historical logs on the primary node is greater than the value of this parameter, the logs are forcibly recycled. Exception: In synchronous commit mode (that is, the value of **synchronous_commit** is not **local** or **off**), if there are connected standby nodes, the primary node retains the logs that meet the

minimum log receiving requirements on the majority of standby nodes. In this case, the number of reserved logs may exceed the value of **max_size_for_xlog_prune**.

3. If any standby node is being built, this parameter does not take effect. All logs of the primary node are retained to prevent build failures due to log recycling.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 2147483647. The unit is KB.

Default value: **256 GB**

17.5.3 Log Replay

recovery_time_target

Parameter description: Specifies the time for a standby server to write and replay logs.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 3600. The unit is s.

0 indicates that log flow control is disabled. A value from **1** to **3600** indicates that a standby server can write and replay logs within the period specified by the value, so that the standby server can quickly assume the primary role. If this parameter is set to a small value, the performance of the primary node is affected. If it is set to a large value, the log flow is not effectively controlled.

Default value: **60**

recovery_max_workers

Parameter description: Specifies the maximum number of concurrent replay threads.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 20

Default value: **4** (For better performance, the default value in tool installation is **4**.)

recovery_parse_workers

Parameter description: Specifies the number of **ParseRedoRecord** threads for the extreme Recovery Time Objective (RTO) feature.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 1 to 16

This parameter can be set to a value greater than 1 only when the extreme RTO feature is enabled. In addition, it must be used together with

recovery_redo_workers. If both **recovery_parse_workers** and **recovery_max_workers** are enabled, the setting of **recovery_parse_workers** prevails and the concurrent replay function is disabled. The extreme RTO feature does not support the hot standby mode. Therefore, **recovery_parse_workers** can be set to a value greater than 1 only when **hot_standby** is set to **off** and **replication_type** to **1**. This feature does not support column-store tables, either. Therefore, disable this feature in a system where column-store tables are used or are to be used. The ultimate RTO does not have flow control. Instead, flow control is determined by the **recovery_time_target** parameter.

Default value: **1**

recovery_redo_workers

Parameter description: Specifies the number of **PageRedoWorker** threads corresponding to each **ParseRedoRecord** thread when the ultimate RTO feature is enabled.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 1 to 8

This parameter must be used together with **recovery_parse_workers**, and it takes effect only when **recovery_parse_workers** is set to a value greater than 0.

Default value: **1**

recovery_parallelism

Parameter description: Specifies the actual number of replay threads. This parameter is read-only.

This parameter is a POSTMASTER parameter and is affected by recovery_max_workers and recovery_parse_workers. If any value is greater than 0, recover_parallelism will be recalculated.

Value range: an integer ranging from 1 to 2147483647

Default value: **1**

enable_page_lsn_check

Parameter description: Specifies whether to enable the data page LSN check. During replay, the current LSN of the data page is checked to see if it is the expected one.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

Default value: **on**

recovery_min_apply_delay

Parameter description: Specifies the replay delay of the standby node.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

- This parameter does not take effect on the primary node. It must be set on the standby node that requires a delay. You are advised to set this parameter on the asynchronous standby node. If the delay is set on the asynchronous standby node, the RTO will be long after the node is promoted to primary.
- The delay time is calculated based on the transaction commit timestamp on the primary server and the current time on the standby server. Therefore, ensure that the clocks of the primary and standby servers are synchronized.
- If the delay time is too long, the disk where the XLOG file is located on the standby node may be full. Therefore, you need to set the delay time based on the disk size.
- Operations without transactions are not delayed.
- After the primary/standby switchover, if the original primary node needs to be delayed, you need to manually set this parameter.
- When **synchronous_commit** is set to **remote_apply**, synchronous replication is affected by the delay. Each commit message is returned only after the replay on the standby server is complete.
- Using this feature also delays **hot_standby_feedback**, which may cause the primary server to bloat, so be careful when using both.
- If a DDL operation (such as DROP or TRUNCATE) that holds an AccessExclusive lock is performed on the primary node, the query operation on the operation object on the standby node will be returned only after the lock is released during the delayed replay of the record on the standby node.
- The MOT table is not supported.

Value range: an integer ranging from 0 to INT_MAX. The unit is ms.

Default value: **0** (no delay added)

redo_bind_cpu_attr

Parameter description: Specifies the core binding operation of the replay thread. Only the **sysadmin** user can access this parameter. This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string of more than 0 characters. The value is case-insensitive.

- **'nobind'**: The thread is not bound to a core.
- **'nodebind: 1, 2'**: Use the CPU cores in NUMA groups 1 and 2 to bind threads.
- **'cpubind: 0-30'**: Use the CPU cores 0 to 30 to bind threads.

Default value: **'nobind'**

17.5.4 Archiving

archive_mode

Parameter description: Specifies whether to archive WALs.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

- When **wal_level** is set to **minimal**, the **archive_mode** parameter is unavailable.
- The archiving function can be enabled on both the synchronous and asynchronous standby nodes. The method of enabling the archiving function is the same as that of enabling the archiving function on a single node. To enable the archiving function, set **archive_mode** to **on** and set **archive_dest** or **archive_command** correctly.
- If the maximum availability mode is not enabled and the standby node is disconnected from the primary node, the primary node cannot send the archiving location to the standby node due to service congestion. As a result, the archiving fails.

Value range: Boolean

- **on** indicates that the archiving is enabled.
- **off** indicates that the archiving is disabled.

Default value: **off**

archive_command

Parameter description: Specifies the command set by the administrator to archive WALs. You are advised to set the archive log path to an absolute path.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

- If both **archive_dest** and **archive_command** are configured, WALs are preferentially saved to the directory specified by **archive_dest**. The command configured by **archive_command** does not take effect.
- Any **%p** in the string is replaced by the absolute path of the file to archive, and any **%f** is replaced by only the file name. (The relative path is relative to the data directory.) Use **%%** to embed an actual **%** character in the command.
- This command returns zero only if it succeeds. The command example is as follows:
  ```
  archive_command = 'cp --remove-destination %p /mnt/server/archivedir/%f'
  ```
- **--remove-destination** indicates that files will be overwritten during the archiving.
- If there are multiple archive commands, write them to the shell script file and set **archive_command** to the command for executing the script. The command example is as follows:
  ```
  -- Assume that multiple commands are as follows:
  test ! -f dir/%f && cp %p dir/%f
  -- The content of the test.sh script is as follows:
  test ! -f dir/$2 && cp $1 dir/$2
  -- The archive command is as follows:
  archive_command='sh dir/test.sh %p %f'
  ```

Value range: a string

Default value: (disabled)

archive_dest

Parameter description: Specifies the path set by the administrator to archive WALs. You are advised to set the archive log path to an absolute path.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

- If both **archive_dest** and **archive_command** are configured, WALs are preferentially saved to the directory specified by **archive_dest**. The command configured by **archive_command** does not take effect.
- If the string is a relative path, it is relative to the data directory. The following is an example:
  ```
  archive_dest = '/mnt/server/archivedir/'
  ```

Value range: a string

Default value: empty

archive_timeout

Parameter description: Specifies the archiving period.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

- The server is forced to switch to a new WAL segment file when the period specified by this parameter has elapsed since the last file switch.
- Archived files that are closed early due to a forced switch are still of the same length as full files. Therefore, a very short **archive_timeout** will bloat the archive storage. You are advised to set **archive_timeout** to **60s**.

Value range: an integer ranging from 0 to 1073741823. The unit is s. **0** indicates that archiving timeout is disabled.

Default value: **0**

archive_interval

Parameter description: Specifies the archiving interval.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

- Log files are forcibly archived when the period specified by this parameter has elapsed.
- Archiving involves I/O operations. Therefore, frequent archiving is not allowed. In addition, the RPO cannot be set to a large value; otherwise, the PITR will be affected. You are advised to use the default value.

Value range: an integer ranging from 1 to 1000. The unit is second.

Default value: **1**

time_to_target_rpo

Parameter description:

In the remote DR mode of dual database instances, set the allowed *time_to_target_rpo* seconds from the time when an exception occurs on the primary database instance to the time when data is archived to the OBS recovery point.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 3600. The unit is s.

In remote DR mode of dual database instances, logs of the primary database instance are archived to OBS. **0** indicates that log flow control is disabled. 1 to 3600 indicates the maximum *time_to_target_rpo* seconds from the time when an exception occurs on the primary database instance to the time when data is archived to the recovery point of OBS. This ensures that the maximum duration of data loss is within the allowed range when the primary database instance breaks down due to a disaster. If this parameter is set to a small value, the performance of the primary node is affected. If it is set to a large value, the log flow is not effectively controlled.

Default value: **0**

17.6 HA Replication

17.6.1 Sending Server

max_wal_senders

Parameter description: Specifies the maximum number of simultaneously running WAL sender processes. The value cannot be greater than or equal to that of **max_connections**.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

wal_level must be set to **archive**, **hot_standby**, or **logical** to allow the connection from standby servers.

Value range: an integer ranging from 0 to 1024. The recommended value range is 8 to 100.

NOTE

This parameter can be set to 0 only when a single DN is used and there is no primary/standby instance.

Default value: 20

wal_keep_segments

Parameter description: Specifies the number of Xlog file segments. Specifies the minimum number of transaction log files stored in the **pg_xlog** directory. The standby node obtains log files from the primary node for streaming replication.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 2 to *INT_MAX*

Default value: **1024**

Setting suggestions:

- During WAL archiving or recovery from a checkpoint on the server, the system may retain more log files than the number specified by **wal_keep_segments**.

- If this parameter is set to an excessively small value, a transaction log may have been overwritten by a new transaction before requested by the standby server. As a result, the request fails and the connection between the primary and standby servers is terminated.
- If the HA system uses asynchronous transmission, increase the value of **wal_keep_segments** when data greater than 4 GB is continuously imported in COPY mode. Take T6000 board as an example. If the data to be imported reaches 50 GB, you are advised to set this parameter to **1000**. You can dynamically restore the setting of this parameter after data import is complete and the WAL synchronization is proper.
- If the synchronous_commit level is lower than LOCAL_FLUSH, you are advised to set this parameter to 1000 when rebuilding the standby node to prevent rebuilding failures caused by primary node log recycling during the rebuilding.

wal_sender_timeout

Parameter description: Specifies the maximum duration that the sending server waits for the WAL reception in the receiver.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

- If the data volume on the primary node is huge, the value of this parameter must be increased for the database rebuilding on a standby node. For example, if the data volume on the primary node reaches 500 GB, you are advised to set this parameter to 600 seconds.
- This parameter cannot be set to a value larger than the value of **wal_receiver_timeout** or the timeout parameter for database rebuilding.

Value range: an integer ranging from 0 to *INT_MAX*. The unit is ms.

Default value: **6s**

max_replication_slots

Parameter description: Specifies the number of log replication slots on the primary node.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 1024. The recommended value range is 8 to 100.

Default value: 20

Setting suggestions:

When HA replication, backup and restoration, and logical decoding are used, you are advised to set this parameter to: Number of current physical replication slots + Number of backup slots + Number of required logical replication slots. If the actual value is smaller than the recommended value, these functions may be unavailable or abnormal.

- Physical replication slots provide an automatic method to ensure that Xlogs are not removed from a primary node before they are received by all the standby nodes and secondary nodes. That is, physical replication slots are used to support primary/standby HA. The number of physical replication slots required by a database is equal to the ratio of standby and secondary nodes to the primary node. For example, if an HA database has 1 primary node, 1 standby node, and 1 secondary node, the number of required physical replication slots will be 2. If an HA database has 1 primary node and 3 standby nodes, the number of required physical replication slots will be 3.
- Backup slot records replication information during backup execution. Full backup and incremental backup correspond to two independent backup slots.
- Currently, primary/standby/secondary deployment is not supported by default.
- Plan the number of logical replication slots as follows:
  - A logical replication slot can carry changes of only one database for decoding. If multiple databases are involved, create multiple logical replication slots.
  - If logical replication is needed by multiple target databases, create multiple logical replication slots in the source database. Each logical replication slot corresponds to one logical replication link.

enable_slot_log

Parameter description: Specifies whether to enable primary/standby synchronization for logical replication slots.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that primary/standby synchronization is enabled for logical replication slots.
- **off** indicates that primary/standby synchronization is disabled for logical replication slots.

Default value: on

max_changes_in_memory

Parameter description: Specifies the maximum number of DML statements cached in memory for a single transaction during logical decoding.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 1 to 2147483647

Default value: **4096**

max_cached_tuplebufs

Parameter description: Specifies the upper limit of the total tuple information cached in the memory during logical decoding. You are advised to set this parameter to a value greater than or equal to twice of **max_changes_in_memory**.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 1 to 2147483647

Default value: **8192**

logical_decode_options_default

Parameter description: Specifies the global default value for unspecified decoding options when logical decoding starts.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Currently, the following logical decoding options are supported: **parallel-decode-num**, **parallel-queue-size**, **max-txn-in-memory**, **max-reorderbuffer-in-memory**, and **exclude-users**. For the meanings of the options, see **Example: Logic Replication Code**.

Value range: a string of key=value characters separated by commas (,), for example, **'parallel-decode-num=4,parallel-queue-size=128,exclude-users=userA'**. An empty string indicates that the default value hardcoded by the program is used.

Default value: empty

NOTICE

The SIGHUP parameter does not affect the started logic decoding process. The options specified by this parameter are used as the default settings for subsequent logic decoding startup, and the settings specified in the startup command are preferentially used.

The **exclude-users** option is different from the logic decoding startup option. You are not allowed to specify multiple blacklisted users.

logical_sender_timeout

Parameter description: Specifies the maximum waiting time for the sender to wait for the receiver to receive logical logs.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 2147483647. The unit is ms.

Default value: **30s**

enable_wal_shipping_compression

Parameter description: Specifies whether to enable cross-database instance log compression in streaming DR mode.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

This parameter applies only to a pair of walsender and walreceiver for cross-database instance transmission in streaming DR and is configured on the primary database instance.

Value range: Boolean

- **true** indicates that cross-database instance log compression is enabled for streaming DR.
- **false** indicates that cross-database instance log compression is disabled for streaming DR.

Default value: **false**

repl_auth_mode

Parameter description: Specifies the validation mode for primary/standby replication and standby node rebuilding.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

- If UUID validation is enabled on the primary node and a non-null repl_uuid validation code is configured, UUID validation must also be enabled on the standby node and the same repl_uuid validation code must be configured on the standby node. Otherwise, requests for log replication between the primary and standby nodes and standby node rebuilding will be rejected by the primary node.
- The SIGHUP parameter can dynamically load new values. The modification does not affect the established primary/standby connection and takes effect for subsequent primary/standby replication requests and primary/standby rebuilding requests.
- It supports the standby node rebuild validation under the Quorum and DCF protocols and the primary/standby replication validation under the Quorum protocol. It does not support primary/standby replication validation under the DCF protocol.
- Authentication between primary and standby database instances is not supported, including primary and standby Dorado instances and DR instances.
- The UUID validation function is used to prevent data crosstalk and pollution caused by incorrect connection between the primary and standby nodes. It is not used for security purposes.
- This parameter cannot be automatically synchronized between the primary and standby nodes.

Value range: enumerated values

- **off**: indicates that UUID validation is disabled.
- **default**: indicates that UUID validation is disabled.
- **uuid**: indicates that UUID validation is enabled.

Default value: **default**

repl_uuid

Parameter description: Specifies the UUID used for primary/standby UUID validation.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

- If UUID validation is enabled on the primary node and a non-null repl_uuid validation code is configured, UUID validation must also be enabled on the standby node and the same repl_uuid validation code must be configured on the standby node. Otherwise, requests for log replication between the primary and standby nodes and standby node rebuilding will be rejected by the primary node.
- The SIGHUP parameter can dynamically load new values. The modification does not affect the established primary/standby connection and takes effect for subsequent primary/standby replication requests and primary/standby rebuilding requests.
- It supports the standby node rebuild validation under the Quorum and DCF protocols and the primary/standby replication validation under the Quorum protocol. It does not support primary/standby replication validation under the DCF protocol.
- Authentication between primary and standby database instances is not supported, including primary and standby Dorado instances and DR instances.
- The UUID validation function is used to prevent data crosstalk and pollution caused by incorrect connection between the primary and standby nodes. It is not used for security purposes.
- This parameter cannot be automatically synchronized between the primary and standby nodes.

Value range: a string. The value is a string of 0 to 63 case-insensitive letters and digits. It is converted to lowercase letters for storage. An empty string indicates that UUID validation is disabled.

Default value: empty

replconninfo1

Parameter description: Specifies the information about the first node to be listened on and authenticated by the current server.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string. An empty string indicates that no information about the first node is configured.

Default value: empty

replconninfo2

Parameter description: Specifies the information about the second node to be listened on and authenticated by the current server.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string. An empty string indicates that no information about the second node is configured.

Default value: empty

replconninfo3

Parameter description: Specifies the information about the third node to be listened on and authenticated by the current server.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string. An empty string indicates that no information about the third node is configured.

Default value: empty

replconninfo4

Parameter description: Specifies the information about the fourth node to be listened on and authenticated by the current server.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string. An empty string indicates that no information about the fourth node is configured.

Default value: empty

replconninfo5

Parameter description: Specifies the information about the fifth node to be listened on and authenticated by the current server.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string An empty string indicates that no information about the fifth node is configured.

Default value: empty

replconninfo6

Parameter description: Specifies the information about the sixth node to be listened on and authenticated by the current server.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string. An empty string indicates that no information about the sixth node is configured.

Default value: empty

replconninfo7

Parameter description: Specifies the information about the seventh node to be listened on and authenticated by the current server.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string. An empty string indicates that no information about the seventh node is configured.

Default value: empty

replconninfo8

Parameter description: Specifies the information about the eighth node to be listened on and authenticated by the current server.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string. An empty string indicates that no information about the eighth node is configured.

Default value: empty

cross_cluster_replconninfo1

Parameter description: Specifies the information about the local first node to be listened on and authenticated across database instances.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string. An empty string indicates that no information about the first node is configured.

Default value: empty

cross_cluster_replconninfo2

Parameter description: Specifies the information about the local second node to be listened on and authenticated across database instances.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string. An empty string indicates that no information about the second node is configured.

Default value: empty

cross_cluster_replconninfo3

Parameter description: Specifies the information about the local third node to be listened on and authenticated across database instances.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string. An empty string indicates that no information about the third node is configured.

Default value: empty

cross_cluster_replconninfo4

Parameter description: Specifies the information about the local fourth node to be listened on and authenticated across database instances.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string. An empty string indicates that no information about the fourth node is configured.

Default value: empty

cross_cluster_replconninfo5

Parameter description: Specifies the information about the local fifth node to be listened on and authenticated across database instances.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string An empty string indicates that no information about the fifth node is configured.

Default value: empty

cross_cluster_replconninfo6

Parameter description: Specifies the information about the local sixth node to be listened on and authenticated across database instances.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string. An empty string indicates that no information about the sixth node is configured.

Default value: empty

cross_cluster_replconninfo7

Parameter description: Specifies the information about the local seventh node to be listened on and authenticated across database instances.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string. An empty string indicates that no information about the seventh node is configured.

Default value: empty

cross_cluster_replconninfo8

Parameter description: Specifies the information about the local eighth node to be listened on and authenticated across database instances.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string. An empty string indicates that no information about the eighth node is configured.

Default value: empty

available_zone

Parameter description: Specifies the region where the local node is located.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string. An empty string indicates that no information about the node is configured.

Default value: empty

enable_availablezone

Parameter description: Specifies whether the local cascaded standby node can connect to standby nodes across AZs.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **true** indicates that the cascaded standby node can only connect to standby nodes in the same AZ.
- **false** indicates that the cascaded standby node can connect to standby nodes across AZs.

Default value: **false**

max_keep_log_seg

Parameter description: Stream control parameter. In logical replication, physical logs are parsed and converted into logical logs locally on the DN. When the number of physical log files that are not parsed is greater than the value of this parameter, stream control is triggered. The value **0** indicates that the stream control function is disabled.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 2147483647

Default value: **0**

17.6.2 Primary Server

synchronous_standby_names

Parameter description: Specifies a comma-separated list of names of potential standby nodes that support synchronous replication.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

- The current synchronous standby node is on the top of the list. If the current synchronous standby node is disconnected, it will be replaced immediately with the next-highest-priority standby node. Name of the next-highest-priority standby node is added to the list.
- The standby node name can be specified by setting the environment variable **PGAPPNAME**.

Value range: a string If this parameter is set to *****, the name of any standby node that provides synchronous replication is matched. The value can be configured in the following format:

- ANY *num_sync* (*standby_name* [, ...]) [, ANY *num_sync* (*standby_name* [, ...])]
- [FIRST] *num_sync* (*standby_name* [, ...])
- *standby_name* [, ...]

NOTE

- In the preceding command, *num_sync* indicates the number of standby nodes that need to wait for responses from the transaction, *standby_name* indicates the name of the standby node, and FIRST and ANY specify the policies for selecting standby nodes for synchronous replication from the listed servers.
- **ANY N (dn_instanceId1, dn_instanceId2,...)** indicates that any *N* host names in the brackets are selected as the name list of standby nodes for synchronous replication. For example, **ANY 1(dn_instanceId1, dn_instanceId2)** indicates that any one of **dn_instanceId1** and **dn_instanceId2** is used as the standby node for synchronous replication.
- **FIRST N (dn_instanceId1, dn_instanceId2, ...)** indicates that the first N primary node names in the brackets are selected as the standby node name list for synchronous replication based on the priority. For example, **FIRST 1 (dn_instanceId1, dn_instanceId2)** indicates that **dn_instanceId1** is selected as the standby node for synchronous replication.
- The meanings of dn_instanceId1, dn_instanceId2, ... are the same as those of FIRST 1 (dn_instanceId1, dn_instanceId2, ...).

If you use the gs_guc tool to set this parameter, perform the following operations:

```
gs_guc reload -Z datanode -N @NODE_NAME@ -D @DN_PATH@ -c "synchronous_standby_names='ANY
NODE 1(dn_instanceId1, dn_instanceId2)'";
```

or

```
gs_guc reload -Z datanode -N @NODE_NAME@ -D @DN_PATH@ -c "synchronous_standby_names='ANY
1(AZ1, AZ2)'";
```

Default value: *

most_available_sync

Parameter description: Specifies whether transactions on the primary node are not blocked due to faults on synchronous standby nodes. For example, if one of the two synchronous standby nodes is faulty and the other is normal, the primary node waits for the normal synchronous standby node instead of being blocked by the faulty synchronous standby node.

For another example, when the quorum protocol is executed, one primary node and three synchronous standby nodes are configured. ANY 2 (node1, node2, and node3) is configured. When node1 and node3 are faulty and node2 is normal, host services are not blocked.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the primary node is not blocked when all synchronous standby nodes are faulty.
- **off** indicates that the primary node is blocked when all synchronous standby nodes are faulty.

Default value: **off**

keep_sync_window

Parameter description: Specifies the delay for entering the maximum availability mode.

- If **most_available_sync** is set to **on**, when synchronous standby nodes are faulty in primary/standby scenarios and the number of configured synchronous standby nodes is insufficient (for details, see the meaning of **synchonous_standby_name**), setting **keep_sync_window** will retain the maximum protection mode within the time window specified by **keep_sync_window**. That is, committing transactions on the primary node is blocked, delay the primary node to enter the maximum availability mode.
- If synchronous standby nodes recover from faults and the number of synchronous standby nodes meets the configuration requirements, transactions are not blocked.
- You are advised to set **keep_sync_window** to 5s. This prevents the monitoring system from incorrectly reporting network instability.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to *INT_MAX*. The unit is second.

- The value **0** indicates that the **keep_sync_window** is not set, that is, the maximum availability mode is entered directly.
- Other values indicate the size of the timeout window.

Default value: **0**

NOTE

Setting this parameter may affect the RPO. If the primary node is faulty within the configured timeout window, the data generated from the time when the primary node is blocked to the time when the primary node is faulty may be lost.

enable_stream_replication

Parameter description: Specifies whether data and logs are synchronized between primary and standby nodes.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

- This parameter is used for performance testing in scenarios where data synchronization to standby nodes is enabled and where it is disabled. If this parameter is set to **off**, tests on abnormal scenarios, such as switchover and faults, cannot be performed to prevent inconsistency between the primary/standby nodes.
- This parameter is a restricted parameter, and you are advised not to set it to **off** in normal service scenarios.
- Currently, primary/standby/secondary deployment is not supported by default.

Value range: Boolean

- **on** indicates that data and log synchronization is enabled.
- **off** indicates that data and log synchronization is disabled.

Default value: **on**

enable_mix_replication

Parameter description: Specifies how WAL files and data are replicated between primary and standby nodes.

This parameter is an INTERNAL parameter. Its default value is **off** and cannot be modified.

NOTICE

- This parameter cannot be modified in normal service scenarios. That is, mixed replication of the WAL files and data pages is disabled.
- Currently, primary/standby/secondary deployment is not supported by default.

Value range: Boolean

- **on** indicates that the WAL file and data page mixed replication mode is enabled.

- **off** indicates that the WAL file and data page mixed replication mode is disabled.

Default value: **off**

vacuum_defer_cleanup_age

Parameter description: Specifies the number of transactions by which **VACUUM** will defer the cleanup of invalid row-store table records, so that **VACUUM** and **VACUUM FULL** do not clean up deleted tuples immediately.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 1000000. **0** means no delay.

Default value: **0**

data_replicate_buffer_size

Parameter description: Specifies the amount of memory used by queues when the sender sends data pages to the receiver. The value of this parameter affects the buffer size used during the replication from the primary node to the standby node.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 4096 to 1072693248. The unit is KB.

Default value: 16 MB (that is, 16384 KB)

walsender_max_send_size

Parameter description: Specifies the size of the WAL or Sender buffers on the primary node.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 8 to *INT_MAX*. The unit is KB.

Default value: **8 MB** (8192 KB)

enable_data_replicate

Parameter description: Specifies how data is synchronized between primary and standby nodes when the data is imported to a row-store table.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the primary and standby nodes synchronize data using data pages when the data is imported to a row-store table. When **replication_type** is set to **1**, this parameter cannot be set to **on**. If this parameter is set to **on** using the GUC tool, its value will be forcibly changed to **off**.

- **off** indicates that the primary and standby nodes synchronize data using Xlogs when the data is imported to a row-store table.

Default value: **off**

ha_module_debug

Parameter description: Specifies the replication status log of a specific data block during data replication.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the status of each data block is recorded in logs during data replication.
- **off** indicates that the status of each data block is not recorded in logs during data replication.

Default value: **off**

enable_incremental_catchup

Parameter description: Specifies the data catchup mode between the primary and standby nodes. Currently, primary/standby/secondary deployment is not supported by default.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the standby node uses the incremental catchup mode. That is, the standby node scans local data files on the standby node to obtain the list of differential data files between the primary and standby nodes and then performs catchup between the primary and standby nodes.
- **off** indicates that the standby node uses the full catchup mode. That is, the standby node scans all local data files on the primary node to obtain the list of differential data files between the primary and standby nodes and then performs catchup between the primary and standby nodes.

Default value: **on**

wait_dummy_time

Parameter description: Specifies the maximum duration for waiting for the secondary nodes to start and send the scanning lists when incremental data catchup is enabled in the database.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 1 to *INT_MAX*. The unit is second.

Default value: **300**

NOTE

- The unit can only be second.
- Currently, primary/standby/secondary deployment is not supported by default.

catchup2normal_wait_time

Parameter description: Specifies the maximum duration for the standby node to catch up with the primary node when **most_available_sync** is enabled in primary/standby scenarios. The value of this parameter is an estimate and may be different from the actual value.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from –1 to 10000. The unit is ms.

- The value **–1** indicates that the primary node is blocked until the data catchup on the standby node is complete.
- The value **0** indicates that the primary node is not blocked during the data catchup on the standby node.
- Other values indicate the maximum duration that the primary node is blocked during the data catchup on the standby node. For example, if this parameter is set to **5000**, the primary node is blocked until the data catchup on the standby node is complete in 5s.

Default value: **–1**

check_sync_standby

Parameter description: Specifies whether to enable the standby node check function. After the **synchronous_standby_names** parameter is correctly configured in the primary/standby scenario, if the synchronous standby node is faulty, the write service on the primary node reports a write failure. This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: on or **off**

- **on** indicates that the standby node check is enabled.
- **off** indicates that the standby node check is disabled.

Default value: **off**

NOTE

- This parameter cannot be synchronized in job work and autonomous transactions. Otherwise, the check may not take effect.
- If the standby node check is not configured for a specified user or session and the standby node is faulty when the forcible synchronization commission mode is enabled, the write operation on a table causes the query of the same table in another user or session to hang. In this case, you need to recover the standby node or manually terminate the hung client.
- The standby node check function cannot be enabled in scenarios (such as VACUUM ANALYZE and gs_clean) where non-write operations trigger log writing. If the standby node does not meet the requirements for synchronizing configurations to the standby node, services will be hung in this scenario. In this case, you need to manually terminate the services.

sync_config_strategy

Parameter description: Specifies the policy for synchronizing configuration files between the primary node and standby node.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: enumerated values

- **all_node**: If this parameter is set to **all_node** for the primary node, the primary node is allowed to proactively synchronize configuration files to all standby nodes. If this parameter is set to **all_node** for a standby node, the standby node is allowed to send synchronization requests to its primary node.
- **only_sync_node**: If this parameter is set to **only_sync_node** for the primary node, the primary node is only allowed to proactively synchronize configuration files to all standby nodes. If this parameter is set to **only_sync_node** for a standby node, the standby node is allowed to send synchronization requests to its primary node.
- **none_node**: If this parameter is set to **none_node** for the primary node, the primary node is not allowed to proactively synchronize configuration files to all standby nodes. If this parameter is set to **none_node** for a standby node, the standby node is not allowed to send synchronization requests to its primary node.

Default value: all_node

hadr_recovery_time_target

Parameter description: Specifies whether the standby database instance completes log writing and replay in streaming DR mode.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 3600. The unit is second.

0 indicates that log flow control is disabled. A value from 1 to 3600 indicates that a standby node can write and replay logs within the period specified by the value of **hadr_recovery_time_target**. This ensures that the logs can be written and replayed within the period specified by the value of **hadr_recovery_time_target**

and the standby database instance can be promoted to primary quickly. If this parameter is set to a small value, the performance of the primary node is affected. If it is set to a large value, the log flow is not effectively controlled.

Default value: **0**

hadr_recovery_point_target

Parameter description: Specifies the RPO time allowed for the standby database instance to flush logs to disks in streaming DR mode.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 3600. The unit is second.

0 indicates that log flow control is disabled. A value from 1 to 3600 indicates that the standby node can flush logs to disks within the period specified by the value of **hadr_recovery_point_target**. This ensures that the log difference between the primary and standby database instances is controlled within the period specified by the value of **hadr_recovery_point_target** during the switchover and the standby database instance can be promoted to primary. If this parameter is set to a small value, the performance of the primary node is affected. If it is set to a large value, the log flow is not effectively controlled.

Default value: **0**

hadr_super_user_record_path

Parameter description: Specifies the path for storing encrypted files of the **hadr_disaster** user in the standby database instance in streaming DR mode. This parameter is a SIGHUP parameter.

Set it based on instructions provided in **Table 10-1**.

Modification suggestion: The value is automatically set by the streaming DR password transfer tool and does not need to be manually added.

Value range: a string

Default value: **NULL**

NOTICE

- In a database instance that contains a primary node, a standby node, and a cascaded standby node, the primary node is a sender relative to the standby node and the standby node is a receiver relative to the primary node.
- The sender actively synchronizes the configuration file to the receiver, and the receiver requests the sender to synchronize the configuration file, which are two independent events, so that the configuration files are synchronized. If you do not want to synchronize configuration files, set this parameter to **none_node** on the receiver. If the sender is a standby node, set this parameter to **none_node** only. If the sender is a primary node, set this parameter to **none_node** when the primary node does not synchronize with any standby node; or set this parameter to **only_sync_node** when the primary node synchronizes with synchronous standby nodes only and does not synchronize with asynchronous standby nodes.
- To be specific, the sender sends a configuration file which directly overwrites the corresponding parameter in the configuration file of the receiver. After the policy for synchronizing configuration files is set, even if you modify configuration parameters of the receiver, the modification does not take effect because the sender immediately overwrites these parameters.
- The following configuration parameters are not synchronized even if the policy for synchronizing configuration files is set: application_name, archive_command, audit_directory, available_zone, comm_control_port, comm_sctp_port, listen_addresses, log_directory, port, replconninfo1, replconninfo2, replconninfo3, replconninfo4, replconninfo5, replconninfo6, replconninfo7, replconninfo8, replconninfo9, replconninfo10, replconninfo11, replconninfo12, replconninfo13, replconninfo14, replconninfo15, replconninfo16, replconninfo17, replconninfo18, ssl, ssl_ca_file, ssl_cert_file, ssl_ciphers, ssl_crl_file, ssl_key_file, ssl_renegotiation_limit, ssl_cert_notify_time, synchronous_standby_names, local_bind_address, perf_directory, query_log_directory, asp_log_directory, streaming_router_port, enable_upsert_to_merge, archive_dest, recovery_min_apply_delay, and sync_config_strategy.

17.6.3 Standby Server

hot_standby

Parameter description: Specifies whether to allow connections and queries on a standby node during its recovery.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

- If this parameter is set to **on**, wal_level must be set to **hot_standby** or higher. Otherwise, the database startup fails.
- In an HA system, **hot_standby** cannot be set to **off**, because this setting can affect other features of the HA system.
- If the **hot_standby** parameter was disabled and the **wal_level** parameter was set to a value lower than **hot_standby**, perform the following operations to ensure that the logs to be replayed on the standby node can be queried on the standby node before enabling the **hot_standby** parameter again:
  1. Change the **wal_level** value of the primary and standby nodes to **hot_standby** or higher, and restart the instances for the change to take effect.
  2. Perform the checkpoint operation on the primary node and query the **pg_stat_get_wal_senders()** function to ensure that the **receiver_replay_location** value of each standby node is the same as the **sender_flush_location** value of the primary node. Ensure that the value adjustment of **wal_level** is synchronized to standby nodes and takes effect, and standby nodes do not need to replay low-level logs.
  3. Set the **hot_standby** parameter of the primary and standby nodes to **on**, and restart the instances for the setting to take effect.

Value range: Boolean

- **on** indicates that connections and queries are allowed on the standby node during the recovery.
- **off** indicates that connections and queries are not allowed on the standby node during the recovery.

Default value: **on**

max_standby_archive_delay

Parameter description: Specifies the wait period before queries on a standby node are canceled when the queries conflict with WAL processing and archiving in hot standby mode.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

–1 indicates that the standby node waits until the conflicting queries are complete.

Value range: an integer ranging from –1 to *INT_MAX*. The unit is ms.

Default value: **3s** (3000 ms)

max_standby_streaming_delay

Parameter description: Specifies the wait period before queries on a standby node are canceled when the queries conflict with WAL data receiving through streaming replication in hot standby mode.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**. If this parameter is set to a large value or the service load is heavy, an error may be reported for waiting for transaction replay and flushing to disks.

NOTICE

–1 indicates that the standby node waits until the conflicting queries are complete.

Value range: an integer ranging from –1 to *INT_MAX*. The unit is ms.

Default value: **3s** (3000 ms)

wal_receiver_status_interval

Parameter description: Specifies the maximum interval for notifying the primary node of the WAL Receiver status.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to INT_MAX. The unit is ms.

Default value: **5s** (5000 ms)

NOTICE

If this parameter is set to **0**, the standby node does not send information, such as the log receiving location, to the primary node. As a result, the transaction commit on the primary node may be blocked, and the switchover may fail. In normal service scenarios, you are not advised to set this parameter to **0**.

hot_standby_feedback

Parameter description: Specifies whether a standby node is allowed to send the result of a query performed on it to the primary node, preventing a query conflict.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the ID of the oldest transaction active on standby nodes will be sent to the primary node.
- **off** indicates that the ID of the oldest transaction active on standby nodes will not be sent to the primary node.

Default value: **off**

NOTICE

If this parameter is set to **on**, VACUUM on the primary node will not clean up tuples modified in transactions later than the oldest transaction active on standby nodes.

Therefore, the performance of the primary node will be affected. If playback conflicts with query on the standby node and a query error is reported, you are advised to increase the value of **max_standby_streaming_delay**.

wal_receiver_timeout

Parameter description: Specifies the maximum wait period for a standby node to receive data from the primary node.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to *INT_MAX*. The unit is ms.

Default value: **6s** (6000 ms)

wal_receiver_connect_timeout

Parameter description: Specifies the timeout period for a standby node to connect to the primary node.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to *INT_MAX*/1000. The unit is s.

Default value: **2s**

wal_receiver_connect_retries

Parameter description: Specifies the maximum attempts that a standby node connects to the primary node

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 1 to *INT_MAX*

Default value: **1**

wal_receiver_buffer_size

Parameter description: Specifies the size of the memory buffer that stores the Xlogs received by the standby and secondary nodes. Currently, primary/standby/secondary deployment is not supported by default.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 4096 to 1047552. The unit is KB.

Default value: **64 MB** (65536 KB)

primary_slotname

Parameter description: Specifies the slot name of the primary node corresponding to a standby node. This parameter is used for the mechanisms to verify the primary-standby relationship and delete WALs.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string

Default value: empty

max_logical_replication_workers

Parameter description: Specifies the maximum number of apply worker threads on the subscriber side.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 262143

Default value: 4

17.7 Query Planning

This section describes the method configuration, cost constants, planning algorithm, and some configuration parameters for the optimizer.

NOTE

Two parameters are involved in the optimizer:

- *INT_MAX* indicates the maximum value of the INT data type. The value is **2147483647**.
- *DBL_MAX* indicates the maximum value of the FLOAT data type.

In addition to customer services, global query planning parameters also affect database O&M and monitoring services, such as WDR generation, scale-out, redistribution, and data import and export.

17.7.1 Optimizer Method Configuration

These configuration parameters provide a crude method of influencing the query plans chosen by the query optimizer. If the default plan chosen by the optimizer for a particular query is not optimal, a temporary solution is to use one of these configuration parameters to force the optimizer to choose a different plan. Better ways include adjusting the optimizer cost constants, manually running **ANALYZE**, increasing the value of the **default_statistics_target** parameter, and increasing the amount of the statistics collected in specific columns using **ALTER TABLE SET STATISTICS**.

enable_broadcast

Parameter description: Controls whether the query optimizer uses the broadcast distribution method when it evaluates the cost of stream.

This parameter is a **USERSET** parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on**: enabled.
- **off**: disabled.

Default value: **on**

NOTE

This parameter does not take effect in the current version.

enable_bitmapscan

Parameter description: Specifies the query optimizer's use of bitmap-scan plan types.

This parameter is a **USERSET** parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on**: enabled.
- **off**: disabled.

Default value: **on**

force_bitmapand

Parameter description: Specifies the query optimizer's use of BitmapAnd plan types.

This parameter is a **USERSET** parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on**: enabled.
- **off**: disabled.

Default value: **off**

enable_hashagg

Parameter description: Specifies the query optimizer's use of Hash aggregation plan types.

This parameter is a **USERSET** parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on**: enabled.
- **off**: disabled.

Default value: **on**

enable_hashjoin

Parameter description: Specifies the query optimizer's use of hash-join plan types.

This parameter is a **USERSET** parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on**: enabled.
- **off**: disabled.

Default value: **on**

enable_indexscan

Parameter description: Specifies the query optimizer's use of index-scan plan types.

This parameter is a **USERSET** parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on**: enabled.
- **off**: disabled.

Default value: **on**

enable_indexonlyscan

Parameter description: Specifies the query optimizer's use of index-only-scan plan types.

This parameter is a **USERSET** parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on**: enabled.
- **off**: disabled.

Default value: **on**

enable_material

Parameter description: Specifies the query optimizer's use of materialization. It is impossible to suppress materialization entirely, but setting this variable to **off** prevents the optimizer from inserting materialized nodes.

This parameter is a **USERSET** parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on**: enabled.
- **off**: disabled.

Default value: **on**

enable_mergejoin

Parameter description: Specifies the query optimizer's use of merge-join plan types.

This parameter is a **USERSET** parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on**: enabled.
- **off**: disabled.

Default value: **off**

enable_nestloop

Parameter description: Specifies whether the query optimizer uses the nested-loop join plan type to fully scan internal tables. It is impossible to suppress nested-loop joins entirely, but setting this variable to **off** encourages the optimizer to choose other methods if available.

This parameter is a **USERSET** parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on**: enabled.
- **off**: disabled.

Default value: **off**

enable_index_nestloop

Parameter description: Specifies whether the query optimizer uses the nested-loop join plan type to scan the parameterized indexes of internal tables.

This parameter is a **USERSET** parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on**: enabled.
- **off**: disabled.

Default value: **on**

enable_seqscan

Parameter description: Specifies the query optimizer's use of sequential scan plan types. It is impossible to suppress sequential scans entirely, but setting this variable to **off** encourages the optimizer to choose other methods if available.

This parameter is a **USERSET** parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on**: enabled.
- **off**: disabled.

Default value: **on**

enable_sort

Parameter description: Specifies the query optimizer's choice of sort methods. It is impossible to suppress explicit sorts entirely, but setting this variable to **off** encourages the optimizer to choose other methods if available.

This parameter is a **USERSET** parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on**: enabled.
- **off**: disabled.

Default value: **on**

enable_tidscan

Parameter description: Specifies the query optimizer's use of Tuple ID (TID) scan plan types.

This parameter is a **USERSET** parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on**: enabled.
- **off**: disabled.

Default value: **on**

enable_kill_query

Parameter description: In CASCADE mode, when a user is deleted, all the objects belonging to the user are deleted. This parameter specifies whether the queries of the objects belonging to the user can be unlocked when the user is deleted.

This parameter is a **SUSET** parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the unlocking is allowed.
- **off** indicates that the unlocking is not allowed.

Default value: **off**

enforce_a_behavior

Parameter description: Specifies the rule matching modes of regular expressions.

This parameter is a **USERSET** parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the A matching rule is used.
- **off** indicates that the POSIX matching rule is used.

Default value: **on**

max_recursive_times

Parameter description: Specifies the maximum number of **WITH RECURSIVE** iterations.

This parameter is a **USERSET** parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to *INT_MAX*

Default value: **200**

enable_vector_engine

Parameter description: Specifies the query optimizer's use of vectorized executor.

This parameter is a **USERSET** parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on**: enabled.
- **off**: disabled.

Default value: **on**

enable_change_hjcost

Parameter description: Specifies whether the optimizer excludes internal table running costs when selecting the Hash Join cost path. If it is set to **on**, tables with a few records and high running costs are more possible to be selected.

This parameter is a **SUSET** parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on**: enabled.

- **off**: disabled.

Default value: **off**

enable_absolute_tablespace

Parameter description: Specifies whether the tablespace can use the absolute path.

This parameter is a **USERSET** parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that an absolute path can be used.
- **off** indicates that an absolute path cannot be used.

Default value: **on**

enable_valuepartition_pruning

Parameter description: Specifies whether the DFS partitioned table is dynamically or statically optimized.

This parameter is a **USERSET** parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the DFS partitioned table is dynamically or statically optimized.
- **off** indicates that the DFS partitioned table is not dynamically or statically optimized.

Default value: **on**

qrw_inlist2join_optmode

Parameter description: Specifies whether to enable inlist-to-join (inlist2join) query rewriting.

This parameter is a **USERSET** parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string

- **disable** indicates that the inlist2join query rewriting is disabled.
- **cost_base** indicates that the cost-based inlist2join query rewriting is enabled.
- **rule_base** indicates that the forcible rule-based inlist2join query rewriting is enabled.
- A positive integer indicates the threshold of inlist2join query rewriting. If the number of elements in the list is greater than the threshold, the rewriting is performed.

Default value: **cost_base**

skew_option

Parameter description: Specifies whether an optimization policy is used.

This parameter is a **USERSET** parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string

- **off** indicates that the policy is disabled.
- **normal** indicates that a radical policy is used. All possible skews are optimized.
- **lazy** indicates that a conservative policy is used. Uncertain skews are ignored.

Default value: **normal**

default_limit_rows

Parameter description: Specifies the default estimated number of limit rows for generating genericplan. If this parameter is set to a positive value, the positive value is used as the estimated number of limit rows. If this parameter is set to a negative value, the negative value is converted to a percentage and used as default estimated value, that is, -5 indicates 5%.

This parameter is a **USERSET** parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a floating point number ranging from –100 to DBL_MAX

Default value: **–10**

check_implicit_conversions

Parameter description: Specifies whether to check candidate index paths generated for index columns that have implicit type conversions in a query.

This parameter is a **USERSET** parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that a check will be performed for candidate index paths generated for index columns that have implicit type conversion in a query.
- **off** indicates that a check will not be performed.

Default value: **off**

cost_weight_index

Parameter description: Specifies the cost weight of index_scan.

This parameter is a **USERSET** parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a floating point number ranging from 1e-10 to 1e+10.

Default value: **1**

try_vector_engine_strategy

Parameter description: Specifies the policy for processing row-store tables using the vectorized executor. By setting this parameter, queries containing row-store tables can be converted to vectorized execution plans for calculation, improving the execution performance of complex queries in AP-like scenarios.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Valid value: enumerated values

- **off** is the default value, which indicates that this function is disabled. That is, row-store tables will not be converted into vectorized execution plans for execution.
- **force**: Queries are forcibly converted to vectorized execution plans for execution no matter whether the base table to be queried is a row-store table, column-store table, or hybrid row-column store table, unless the query type or expression is not supporte by the vectorized executor. In this case, the performance may deteriorate in different query scenarios.
- **optimal**: On the basis of **force**, the optimizer determines whether to convert a query statement into a vectorized execution plan based on the query complexity to avoid performance deterioration after the conversion.

Default value: **off**

17.7.2 Optimizer Cost Constants

This section describes the optimizer cost constants. The cost variables described here are measured on an arbitrary scale. Only their relative values matter, therefore scaling them all up or down by the same factor will result in no change in the optimizer's choices. By default, these cost variables are based on the cost of sequential page fetches, that is, **seq_page_cost** is conventionally set to **1.0** and the other cost variables are set with reference to the parameter. However, you can use a different scale, such as actual execution time in milliseconds.

seq_page_cost

Parameter description: Specifies the optimizer's estimated cost of a disk page fetch that is part of a series of sequential fetches.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a floating point number ranging from 0 to *DBL_MAX*

Default value: **1**

random_page_cost

Parameter description: Specifies the optimizer's estimated cost of an out-of-sequence disk page fetch.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

Although the server allows you to set **random_page_cost** to a value less than that of **seq_page_cost**, it is not physically sensitive to do so. However, setting them equal makes sense if the database is entirely cached in RAM, because in that case there is no penalty for fetching pages out of sequence. Also, in a heavily-cached database you should lower both values relative to the CPU parameters, since the cost of fetching a page already in RAM is much smaller than it would normally be.

Value range: a floating point number ranging from 0 to *DBL_MAX*

Default value: 4

NOTE

- This value can be overwritten for tables and indexes in a particular tablespace by setting the tablespace parameter of the same name.
- Reducing this value relative to **seq_page_cost** will cause the system to prefer index scans and raising it will make index scans relatively more expensive. You can increase or decrease both values together to change the disk I/O costs relative to CPU costs.

cpu_tuple_cost

Parameter description: Specifies the optimizer's estimated cost of processing each row during a query.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a floating point number ranging from 0 to *DBL_MAX*

Default value: **0.01**

cpu_index_tuple_cost

Parameter description: Specifies the optimizer's estimated cost of processing each index entry during an index scan.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a floating point number ranging from 0 to *DBL_MAX*

Default value: 0.005

cpu_operator_cost

Parameter description: Specifies the optimizer's estimated cost of processing each operator or function executed during a query.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a floating point number ranging from 0 to *DBL_MAX*

Default value: **0.0025**

effective_cache_size

Parameter description: Specifies the optimizer's assumption about the effective size of the disk cache that is available to a single query.

When setting this parameter you should consider both GaussDB's shared buffers and the kernel's disk cache. Also, take into account the expected number of concurrent queries on different tables, since they will have to share the available space.

This parameter has no effect on the size of shared memory allocated by GaussDB. It is used only for estimation purposes and does not reserve kernel disk cache. The value is in the unit of disk page. Usually the size of each page is 8192 bytes.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 1 to 2147483647. The unit is 8 KB.

A value greater than the default one makes index scans more likely to be used, whereas a value less than the default one makes sequential scans more likely to be used.

Default value:

180 GB (128-core CPU/1024-GB memory, 104-core CPU/1024-GB memory, 96-core CPU/1024-GB memory); **135 GB** (96-core CPU/768-GB memory); **90 GB** (64-core CPU/512-GB memory); **80 GB** (60-core CPU/480-GB memory); **40 GB** (32-core CPU/256-GB memory); **18 GB** (16-core CPU/128-GB memory); **8 GB** (8-core CPU/64-GB memory); **4 GB** (4-core CPU/32-GB memory); **2 GB** (4-core CPU/16-GB memory)

allocate_mem_cost

Parameter description: Specifies the query optimizer's estimated cost of creating a hash table for memory space using hash join. This parameter is used for optimization when the hash join estimation is inaccurate.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a floating point number ranging from 0 to *DBL_MAX*

Default value: **0**

17.7.3 Genetic Query Optimizer

This section describes parameters related to genetic query optimizer. The genetic query optimizer (GEQO) is an algorithm that plans queries by using heuristic searching. This algorithm reduces planning time for complex queries and the costs of producing plans are sometimes inferior to those found by the normal exhaustive-search algorithm.

geqo

Parameter description: Specifies whether to enable the genetic query optimization.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

It is best not to turn it off in execution. **geqo_threshold** provides more subtle control of GEQO.

Value range: Boolean

- **on** indicates that the genetic query optimization is enabled.
- **off** indicates that the genetic query optimization is disabled.

Default value: **on**

geqo_threshold

Parameter description: Specifies the number of **FROM** items. Genetic query optimization is used to plan queries when the number of statements executed is greater than this value.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

- For simpler queries, it is best to use the regular, exhaustive-search planner; but for queries with many tables, it is better to use GEQO to manage the queries.
- A **FULL OUTER JOIN** construct counts as only one **FROM** item.

Value range: an integer ranging from 2 to *INT_MAX*

Default value: **12**

geqo_effort

Parameter description: Controls the trade-off between planning time and query plan quality in GEQO.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

geqo_effort does not do anything directly. This parameter is only used to compute the default values for the other variables that influence GEQO behavior. If you prefer, you can manually set the other parameters instead.

Value range: an integer ranging from 1 to 10

NOTICE

Larger values increase the time spent in query planning, but also increase the probability that an efficient query plan is chosen.

Default value: 5

geqo_pool_size

Parameter description: Controls the pool size used by GEQO, that is, the number of individuals in the genetic population.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to *INT_MAX*

NOTICE

The value of this parameter must be at least **2**, and useful values are typically from **100** to **1000**. If this parameter is set to **0**, GaussDB selects a proper value based on **geqo_effort** and the number of tables.

Default value: 0

geqo_generations

Parameter description: Specifies the number of iterations of the GEQO.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to *INT_MAX*

NOTICE

The value of this parameter must be at least **1**, and useful values are typically from **100** to **1000**. If it is set to **0**, a suitable value is chosen based on **geqo_pool_size**.

Default value: 0

geqo_selection_bias

Parameter description: Specifies the selection bias used by GEQO. The selection bias is the selective pressure within the population.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a floating point number ranging from 1.5 to 2.0

Default value: **2**

geqo_seed

Parameter description: Specifies the initial value of the random number generator used by GEQO to select random paths through the join order search space.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a floating point number ranging from 0.0 to 1.0

NOTICE

Varying the value changes the set of join paths explored, and may result in a better or worse best path being found.

Default value: **0**

17.7.4 Other Optimizer Options

explain_dna_file

Parameter description: Sets **explain_perf_mode** to **run** to export object files in CSV format.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

The value of this parameter must be an absolute path plus a file name with the extension **.csv**.

Value range: a string

Default value: **empty**

explain_perf_mode

Parameter description: Specifies the display format of the **explain** command.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: normal, **pretty**, **summary**, and **run**

- **normal** indicates that the default printing format is used.
- **pretty** indicates a new format improved by using GaussDB. The new format contains a plan node ID, directly and effectively analyzing performance.

- **summary** indicates that the analysis result on this information is printed in addition to the printed information specified by **pretty**.
- **run** indicates that the system exports the printed information specified by **summary** as a CSV file for further analysis.

Default value: pretty (In the current version, only the value **normal** takes effect. If the value is not **normal**, **normal** is still displayed.)

analysis_options

Parameter description: Specifies whether to enable function options in the corresponding options to use the corresponding location functions, including data verification and performance statistics. For details, see the options in the value range.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string

- **LLVM_COMPILE** indicates that the codegen compilation time of each thread is displayed on the explain performance page. The current feature is a lab feature. Contact Huawei technical support before using it.
- **HASH_CONFLICT** indicates that the log in the **pg_log** directory of the database node process displays the statistics of the hash table, including the hash table size, hash link length, and hash conflict.
- **STREAM_DATA_CHECK** indicates that a CRC check is performed on data before and after network data transmission.

Default value: **ALL,on(),off(LLVM_COMPILE,HASH_CONFLICT,STREAM_DATA_CHECK)**, which indicates that no location function is enabled.

cost_param

Parameter description: Specifies use of different estimation methods in specific customer scenarios, allowing estimated values approximating to onsite values. This parameter can control various methods simultaneously by performing AND (&) on the bit of each method. A method is selected if the result value is not **0**.

When **cost_param & 1** is set to a value other than **0**, an improved mechanism is used for connecting the selection rate of non-equi-joins. This method is more accurate for estimating the selection rate of joins between two identical tables. At present, **cost_param & 1=0** is not used. That is, a better formula is selected for calculation.

When **cost_param & 2** is set to a value other than **0**, the selection rate is estimated based on multiple filter criteria. The lowest selection rate among all filter criteria, but not the product of the selection rates for two tables under a specific filter criterion, is used as the total selection rate. This method is more accurate when a close correlation exists between the columns to be filtered.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to *INT_MAX*

Default value: **0**

enable_partitionwise

Parameter description: Specifies whether to select an intelligent algorithm for joining partition tables.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that an intelligent algorithm is selected.
- **off** indicates that an intelligent algorithm is not selected.

Default value: **off**

rewrite_rule

Parameter description: Specifies the rewriting rule for enabled optional queries. Some query rewrite rules are optional. Enabling them cannot always improve the query efficiency. In a specific customer scenario, you can set the query rewriting rules through this GUC parameter to achieve optimal query efficiency.

This parameter can control the combination of query rewriting rules, for example, there are override rules: rule1, rule2, rule3, and rule4. You can perform the following settings:

```
set rewrite_rule=rule1;          -- Enable query rewriting rule rule1
set rewrite_rule=rule2, rule3;     -- Enable the query rewriting rules rule2 and rule3
set rewrite_rule=none;          -- Disable all optional query rewriting rules
```

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string

- **none**: Does not use any optional query rewriting rules.
- **lazyagg**: Uses the Lazy Agg query rewriting rules for eliminating aggregation operations in subqueries.
- **magicset**: Uses the Magic Set query rewriting rules delivered from the main query to the subquery.
- **uniquecheck**: Uses the Unique Check query rewriting rules. Optimize the subquery statements in target columns without agg and check whether the number of returned rows is 1.
- **intargetlist**: Uses the In Target List query rewriting rules (subquery optimization in the target column).
- **predpushnormal**: Uses the Predicate Push query rewriting rule (push the predicate condition to the subquery).
- **predpushforce**: Uses the Predicate Push query rewriting rules. Push down predicate conditions to subqueries and use indexes as much as possible for acceleration.
- **predpush**: Selects the optimal plan based on the cost in **predpushnormal** and **predpushforce**.

- **disable_pullup_expr_sublink**: Disables optimizers to pull up expr_sublink. For details about sublink classification and pullup principles, see **Optimizing Subqueries**.

Default value: **magicset**

enable_pbe_optimization

Parameter description: Specifies whether the optimizer optimizes the query plan for statements executed in Parse Bind Execute (PBE) mode.

This parameter is a SUSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the optimizer optimizes the query plan of the PBE statement.
- **off** indicates that the optimizer does not optimize the query plan of the PBE statement.

Default value: **on**

enable_global_plancache

Parameter description: Specifies whether to share the cache of the PBE query execution plan. Enabling this function can reduce the memory usage of database nodes in high concurrency scenarios. The current feature is a lab feature. Contact Huawei technical support before using it.

When **enable_global_plancache** is enabled, the default value of **local_syscache_threshold** is greater than or equal to 16 MB to ensure that GPC takes effect. If the value of **local_syscache_threshold** is less than 16 MB, set it to 16 MB. If the value is greater than 16 MB, do not change it.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the execution plan of the PBE query is shared in the cache.
- **off** indicates that the execution plan of the PBE query is not shared in the cache.

Default value: **off**

gpc_clean_timeout

Parameter description: When **enable_global_plancache** is set to **on**, if a plan in the shared plan list is not used within the period specified by **gpc_clean_timeout**, the plan will be deleted. This parameter is used to control the retention period of a shared plan that is not used. The current feature is a lab feature. Contact Huawei technical support before using it.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 300 to 86400

- The unit is second.

Default value: **1800**, that is, 30 minutes

enable_global_stats

Parameter description: Specifies the current statistics collection mode, which can be global statistics collection or single-node statistics collection. By default, the global statistics collection mode is used. If this parameter is set to **off**, the statistics of the first node in the database are collected by default. In this case, the quality of the generated query plan may be affected. However, the information collection performance is optimal. Therefore, exercise caution when setting this parameter.

This parameter is a SUSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** or **true** indicates the global statistics mode.
- **off** or **false** indicates the database node statistics.

Default value: **on**

enable_opfusion

Parameter description: Specifies whether to optimize simple addition, deletion, modification, and query operations.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

The restrictions on simple query are as follows:

- Only indexscan and indexonlyscan are supported, and the filter criteria of all WHERE statements are on indexes.
- Only single tables can be added, deleted, modified, and queried. Join and using are not supported.
- Only row-store tables are supported. Partitioned tables and tables with triggers are not supported.
- Information statistics features such as active sql and QPS are not supported.
- Tables that are being scaled out or in are not supported.
- The system column cannot be queried or modified.
- Only simple **SELECT** statements are supported. For example:
  ```
  SELECT c3 FROM t1 WHERE c1 = ? and c2 =10;
  ```
  Only columns in the target table can be queried. Columns **c1** and **c2** are index columns, which can be followed by constants or parameters. You can use **for update**.
- Only simple **INSERT** statements are supported. For example:
  ```
  INSERT INTO t1 VALUES (?,10,?);
  ```
  Only one **VALUES** is supported. The type in **VALUES** can be a constant or a parameter. **RETURNING** is not supported.

- Only simple **DELETE** statements are supported. For example:

```
DELETE FROM t1 WHERE c1 = ? and c2 = 10;
```

  Columns **c1** and **c2** are index columns, which can be followed by constants or parameters.

- Only simple **UPDATE** statements are supported. For example:

```
UPDATE t1 SET c3 = c3+? WHERE c1 = ? and c2 = 10;
```

  The values modified in column **c3** can be constants, parameters, or a simple expression. Columns **c1** and **c2** are index columns, which can be followed by constants or parameters.

Value range: Boolean

- **on**: enabled.
- **off**: disabled.

Default value: **on**

enable_partition_opfusion

Parameter description: If this parameter is enabled when the **enable_opfusion** parameter is enabled, the simple query of the partitioned table can be optimized to improve the SQL execution performance. When the **enable_global_plancache** parameter is enabled, this parameter does not take effect even if it is set to **on**.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on**: enabled.
- **off**: disabled.

Default value: **off**

sql_beta_feature

Parameter description: Specifies the SQL engine's optional beta features to be enabled, including optimization of row count estimation and query equivalence estimation.

These optional features provide optimization for specific scenarios, but performance deterioration may occur in some scenarios for which testing is not performed. In a specific customer scenario, you can set the query rewriting rules through this GUC parameter to achieve optimal query efficiency.

This parameter determines the combination of the SQL engine's beta features, for example, feature1, feature2, feature3, and feature4. You can perform the following settings:

```
-- Enable beta feature feature1 of the SQL engine.
set sql_beta_feature=feature1;
-- Enable beta features feature2 and feature3 of the SQL engine.
set sql_beta_feature=feature2,feature3;
-- Disable all optional beta features of the SQL engine.
set sql_beta_feature=none;
```

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string

- **none**: None of the beta optimizer features is used.
- **sel_semi_poisson**: Uses poisson distribution to calibrate the equivalent semi-join and anti-join selection rates.
- **sel_expr_instr**: Uses the matching row count to provide more accurate estimation for **instr(col, 'const') > 0, = 0, = 1**.
- **param_path_gen**: Generates more possible parameterized paths.
- **rand_cost_opt**: Optimizes the random read cost of tables that have a small amount of data.
- **param_path_opt**: Uses the bloat ratio of the table to optimize the analysis information of indexes.
- **page_est_opt**: Optimizes the **relpages** estimation for the analysis information of non-column-store table indexes.
- **no_unique_index_first**: Disables optimization of the primary key index scanning path first.
- **join_sel_with_cast_func**: Type conversion functions are supported when the number of join rows is estimated.
- **canonical_pathkey**: after regularization pathkey is generated. (**pathkey**: a set of ordered key values of data.)
- **index_cost_with_leaf_pages_only**: Considers index leaf nodes when the index cost is estimated.
- **partition_opfusion**: Enables partitioned table optimization.
- **a_style_coerce**: Enables the Decode type conversion rule to be compatible with O. For details, see **Type Resolution for CASE in ORA Compatibility Type**.
- **partition_fdw_on**: SQL statements can be created for Postgres foreign tables based on partitioned tables.
- **predpush_same_level**: Enables the **predpush** hint to control parameterized paths at the same layer.
- **enable_plsql_smp**: Enables parallel execution of queries in stored procedures. Currently, only one query can be executed in parallel at a time, and no parallel execution plan is generated for cursor-related operations, autonomous transactions, and queries in exceptions.
- **disable_bitmap_cost_with_lossy_pages**: Disables the computation of the cost of lossy pages in the bitmap path cost.

Default value: "sel_semi_poisson,sel_expr_instr,rand_cost_opt,param_path_opt,page_est_opt"

ngram_gram_size

Parameter description: Specifies the length of the ngram parser segmentation.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 1 to 4

Default value: **2**

ngram_grapsymbol_ignore

Parameter description: Specifies whether the ngram parser ignores graphical characters.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on**: The ngram parser ignores graphical characters.
- **off**: The ngram parser does not ignore graphical characters.

Default value: **off**

ngram_punctuation_ignore

Parameter description: Specifies whether the ngram parser ignores punctuations.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on**: The ngram parser ignores punctuations.
- **off**: The ngram parser does not ignore punctuations.

Default value: **on**

default_statistics_target

Parameter description: Specifies the default statistics target for table columns without a column-specific target set via **ALTER TABLE SET STATISTICS**. If this parameter is set to a positive number, it indicates the number of samples of statistics information. If this parameter is set to a negative number, percentage is used to set the statistic target. The negative number converts to its corresponding percentage, for example, -5 means 5%.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from –100 to 10000

NOTICE

- A larger positive number than the default value increases the time required to do **ANALYZE**, but might improve the quality of the optimizer's estimates.
- Changing settings of this parameter may result in performance deterioration. If query performance deteriorates, you can:
  1. Restore to the default statistics.
  2. Use hints to force the optimizer to use the optimal query plan. For details, see **Hint-based Tuning**.
- If this parameter is set to a negative value, the number of samples is greater than or equal to 2% of the total data volume, and the number of records in user tables is less than 1.6 million, the time taken by running **ANALYZE** will be longer than when this parameter uses its default value.
- If this parameter is set to a negative value, the auto-analyze function is disabled.

Default value: **100**

constraint_exclusion

Parameter description: Specifies the query optimizer's use of table constraints to optimize queries.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: enumerated values

- **on** indicates that constraints for all tables are examined.
- **off** indicates that constraints for any table are not examined.
- **partition** indicates that only constraints for inheritance child tables and **UNION ALL** subqueries are examined.

NOTICE

When **constraint_exclusion** is set to **on**, the optimizer compares query conditions with the table's **CHECK** constraints, and omits scanning tables for which the conditions contradict the constraints.

Default value: **partition**

NOTE

Currently, **constraint_exclusion** is enabled by default only for cases that are often used to implement table partitioning. Turning this feature on for all tables imposes extra planning on simple queries, and provides no benefit for simple queries. If you have no partitioned tables, set it to **off**.

cursor_tuple_fraction

Parameter description: Specifies the optimizer's estimated fraction of a cursor's rows that are retrieved.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a floating point number ranging from 0.0 to 1.0

NOTICE

Smaller values of this setting bias the optimizer towards using **fast start** plans for cursors, which will retrieve the first few rows quickly while perhaps taking a long time to fetch all rows. Larger values put more emphasis on the total estimated time. At the maximum setting of **1.0**, cursors are planned exactly like regular queries, considering only the total estimated time and how soon the first rows might be delivered.

Default value: **0.1**

from_collapse_limit

Parameter description: Specifies whether the optimizer merges sub-queries into upper queries based on the resulting FROM list. The optimizer merges sub-queries into upper queries if the resulting FROM list would have no more than this many items.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 1 to *INT_MAX*

NOTICE

Smaller values reduce planning time but may lead to inferior execution plans.

Default value: **8**

join_collapse_limit

Parameter description: Specifies whether the optimizer rewrites **JOIN** constructs (except **FULL JOIN**) into lists of **FROM** items based on the number of the items in the result list.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 1 to *INT_MAX*

NOTICE

- Setting this parameter to **1** prevents join reordering. As a result, the join order specified in the query will be the actual order in which the relations are joined. The query optimizer does not always choose the optimal join order. Therefore, advanced users can temporarily set this variable to **1**, and then specify the join order they desire explicitly.
- Smaller values reduce planning time but lead to inferior execution plans.

Default value: 8

plan_mode_seed

Parameter description: This is a commissioning parameter. Currently, it supports only **OPTIMIZE_PLAN** and **RANDOM_PLAN**. The value **0** (for **OPTIMIZE_PLAN**) indicates the optimized plan using the dynamic planning algorithm. Other values are for **RANDOM_PLAN**, which indicates that the plan is randomly generated. **–1** indicates that users do not specify the value of the seed identifier. In this case, the optimizer generates a random integer from **1** to **2147483647** and a random execution plan based on the generated integer. A value from **1** to **2147483647** is regarded as the seed identifier, based on which the optimizer generates a random execution plan.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from –1 to 2147483647

Default value: 0

NOTICE

- If **plan_mode_seed** is set to **RANDOM_PLAN**, the optimizer generates a random execution plan that may not be the optimal one. Therefore, to guarantee the query performance, the default value **0** is recommended during upgrade, scale-out, scale-in, and O&M.
- If this parameter is not set to **0**, the specified plan hint will not be used.

hashagg_table_size

Parameter description: Specifies the hash table size during the execution of the HASH JOIN operation.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to *INT_MAX*/2

Default value: 0

enable_codegen

Parameter description: Specifies whether code optimization is enabled. Currently, the code optimization uses the LLVM optimization. The current feature is a lab feature. Contact Huawei technical support before using it.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that code optimization is enabled.
- **off** indicates that code optimization is disabled.

NOTICE

Currently, the LLVM optimization only supports the vectorized executor feature. You are advised to disable this parameter in other scenarios.

Default value: **off**

codegen_strategy

Parameter description: Specifies the codegen optimization strategy that is used when an expression is converted to be codegen-based. The current feature is a lab feature. Contact Huawei technical support before using it.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: enumerated values

- **partial** indicates that even if functions that are not codegen-based exist in an expression, you can still call the LLVM dynamic optimization strategy by using the entire codegen framework of the expression.
- **pure** indicates that only when all functions in an expression can be codegen-based, the LLVM dynamic optimization strategy can be called.

NOTICE

In the scenario where query performance reduces after the codegen function is enabled, you can set this parameter to **pure**. In other scenarios, do not change the default value **partial** of this parameter.

Default value: **partial**

enable_codegen_print

Parameter description: Specifies whether the LLVM IR function can be printed in logs. The current feature is a lab feature. Contact Huawei technical support before using it.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the LLVM IR function can be printed in logs.
- **off** indicates that the LLVM IR function cannot be printed in logs.

Default value: **off**

codegen_cost_threshold

Parameter description: The LLVM compilation takes some time to generate executable machine code. Therefore, LLVM compilation is beneficial only when the actual execution cost is more than the sum of the code required for generating machine code and the optimized execution cost. This parameter specifies a threshold. If the estimated execution cost exceeds the threshold, LLVM optimization is performed. The current feature is a lab feature. Contact Huawei technical support before using it.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 2147483647

Default value: **10000**

enable_bloom_filter

Parameter description: Specifies whether the BloomFilter optimization can be used. This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the BloomFilter optimization can be used.
- **off** indicates that the BloomFilter optimization cannot be used.

Default value: **on**

enable_extrapolation_stats

Parameter description: Specifies whether the extrapolation logic is used for data of DATE type based on historical statistics. The logic can increase the accuracy of estimation for tables whose statistics are not collected in time, but will possibly provide an overlarge estimation due to incorrect extrapolation. Enable the logic only in scenarios where the data of DATE type is periodically inserted. This parameter is a SUSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the extrapolation logic is used for data of DATE type based on historical statistics.
- **off** indicates that the extrapolation logic is not used for data of DATE type based on historical statistics.

Default value: **off**

autoanalyze

Parameter description: Specifies whether to automatically collect statistics on tables that have no statistics when a plan is generated. **autoanalyze** cannot be used for foreign or temporary tables. To collect statistics, manually perform the ANALYZE operation. If an exception occurs in the database during the execution of autoanalyze on a table, after the database is recovered, the system may still prompt you to collect the statistics of the table when you run the statement again. In this case, manually perform the ANALYZE operation on the table to synchronize statistics. This parameter is a SUSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the table statistics are automatically collected.
- **off** indicates that the table statistics are not automatically collected.

Default value: **off**

enable_analyze_check

Parameter description: Checks whether statistics were collected about tables whose **reltuples** and **relpages** are displayed as **0** in **pg_class** during plan generation.

This parameter is a SUSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the tables will be checked.
- **off** indicates that the tables will not be checked.

Default value: **off**

enable_sonic_hashagg

Parameter description: Specifies whether to use the hash aggregation operator designed for column-oriented hash tables when certain constraints are met.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the hash aggregation operator designed for column-oriented hash tables is used when certain constraints are met.
- **off** indicates that the hash aggregation operator designed for column-oriented hash tables is not used.

NOTE

- When the hash aggregation operator designed for column-oriented hash tables is used, the memory usage of the query can be reduced. However, in scenarios when **enable_codegen** is set to **on** and the performance is significantly improved, the performance of the operator may deteriorate.
- If **enable_sonic_hashagg** is set to **on**, when certain constraints are met, the hash aggregation operator designed for column-oriented hash tables is used and its name is displayed as **Sonic Hash Aggregation** in the output of the Explain Analyze/Performance operation. When the constraints are not met, the operator name is displayed as **Hash Aggregation**. For details, see **Description**.

Default value: **on**

enable_sonic_hashjoin

Parameter description: Specifies whether to use the hash join operator designed for column-oriented hash tables when certain constraints are met.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the hash join operator designed for column-oriented hash tables is used when certain constraints are met.
- **off** indicates that the hash join operator designed for column-oriented hash tables is not used.

NOTE

- Currently, the parameter can be used only for Inner Join.
- If **enable_sonic_hashjoin** is enabled, the memory usage of query using the Hash Inner operator can be reduced. However, in scenarios where the code generation technology can significantly improve performance, the performance of the operator may deteriorate.
- If **enable_sonic_hashjoin** is set to **on**, when certain constraints are met, the hash join operator designed for column-oriented hash tables is used and its name is displayed as **Sonic Hash Join** in the output of the Explain Analyze/Performance operation. When the constraints are not met, the operator name is displayed as **Hash Join**. For details, see **Description**.

Default value: **on**

enable_sonic_optspill

Parameter description: Specifies whether to optimize the number of files to be written to disks for the Hash Join operator designed for column-oriented hash tables. If this parameter is set to **on**, the number of files written to disks does not increase significantly when the Hash Join operator writes a large number of files to disks.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the optimization is enabled.

- **off** indicates that the optimization is disabled.

Default value: **on**

log_parser_stats

Parameter description: Specifies whether the optimizer outputs the performance logs of the parser module. (The current feature is a lab feature. Contact Huawei technical support before using it.)

This parameter is a SUSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on**: enabled.
- **off**: disabled.

Default value: **off**

log_planner_stats

Parameter description: Specifies whether the optimizer outputs the performance logs of the planner module. (The current feature is a lab feature. Contact Huawei technical support before using it.)

This parameter is a SUSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on**: enabled.
- **off**: disabled.

Default value: **off**

log_executor_stats

Parameter description: Specifies whether the optimizer outputs the performance logs of the executor module. (The current feature is a lab feature. Contact Huawei technical support before using it.)

This parameter is a SUSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on**: enabled.
- **off**: disabled.

Default value: **off**

log_statement_stats

Parameter description: Specifies whether the optimizer outputs the performance logs of a statement. (The current feature is a lab feature. Contact Huawei technical support before using it.)

This parameter is a SUSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on**: enabled.
- **off**: disabled.

Default value: **off**

plan_cache_mode

Parameter description: Specifies the policy for generating an execution plan in the **prepare** statement.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: enumerated values

- **auto** indicates that the **custom plan** or **generic plan** is selected by default.
- **force_generic_plan** indicates that the **generic plan** is forcibly used.
- **force_custom_plan** indicates that the **custom plan** is forcibly used.

NOTE

- This parameter is valid only for the **prepare** statement. It is used when the parameterized field in the **prepare** statement has severe data skew.
- **custom plan** is a plan generated after you run the **prepare** statement where parameters in the **execute** statement is embedded. The **custom plan** generates a plan based on specific parameters in the **execute** statement. This scheme generates a preferred plan based on specific parameters each time and has good execution performance. The disadvantage is that the plan needs to be regenerated before each execution, resulting in a large amount of repeated optimizer overhead.
- **generic plan** is a plan generated after you run the **prepare** statement. The plan policy binds parameters to the plan when you run the **execute** statement and execute the plan. The advantage of this scheme is that repeated optimizer overheads can be avoided in each execution. The disadvantage is that the plan may not be optimal when data skew occurs for the bound parameters and may result in poor plan execution performance.

Default value: **auto**

enable_hypo_index

Parameter description: Determines whether the optimizer considers virtual indexes when executing the **EXPLAIN** command.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on**: enabled.
- **off**: disabled.

Default value: **off**

enable_force_vector_engine

Parameter description: Specifies whether to forcibly generate vectorized execution plans for a vectorized execution operator if the operator's child node is a non-vectorized operator. When this parameter is set to **on**, vectorized execution plans are forcibly generated.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that vectorized operators are forcibly generated.
- **off** indicates that the vectorized operator optimizer determines whether to perform vectorization.

Default value: **off**

enable_auto_explain

Parameter description: Specifies whether to enable the function of automatically printing execution plans. This parameter is used to locate slow stored procedures or slow queries and is valid only for the currently connected primary database node.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean. The value **true** indicates that the function is enabled, and the value **false** indicates that the function is disabled.

Default value: **false**

auto_explain_level

Parameter description: Specifies the log level for automatically printing execution plans.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: enumerated type. The value can be **LOG** or **NOTICE**. **LOG** indicates that the execution plan is printed in logs. **NOTICE** indicates that the execution plan is printed in notification mode.

Default value: **LOG**

auto_explain_log_min_duration

Parameter description: Specifies how long execution plans are automatically printed for. Plans can be printed only when the time required to execute the plans is greater than the value of **auto_explain_log_min_duration**.

This parameter is a **USERSET** parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 2147483647. The unit is ms.

- **0**: All executed plans are generated.
- **3000**: All execution plans will be generated after the execution of a statement takes more than 3000 ms.

Default value: **0**

query_dop

Parameter description: Specifies the user-defined degree of parallelism (DOP). This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 1 to 64 If the fixed SMP function is enabled, the system uses the fixed DOP.

NOTE

After enabling concurrent queries, ensure you have sufficient CPU, memory, and network to achieve the optimal performance.

Default value: **1**

enable_startwith_debug

Parameter description: Specifies whether to enable the **start with** or **connect by** parameter for debugging. If this parameter is enabled, information about all tail columns related to the **start with** feature is displayed.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean. The value **true** indicates that the function is enabled, and the value **false** indicates that the function is disabled.

Default value: **false**

17.8 Error Reporting and Logging

17.8.1 Logging Destination

log_destination

Parameter description: GaussDB supports several methods of logging server messages. Set this parameter to a list of desired log destinations separated by commas. (For example, log_destination="stderr,csvlog")

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string

The valid values are **stderr**, **csvlog**, **syslog**, and **eventlog**.

- **stderr** indicates that logs are printed to the screen.

- **csvlog** indicates that logs are output in comma separated value (CSV) format. The prerequisite for generating logs in CSV format is that **logging_collector** must be set to **on**. For details, see **Using CSV Log Output**.
- **syslog** indicates that logs are recorded using the syslog of the OS. GaussDB can record logs using syslog from **LOCAL0** to **LOCAL7**. For details, see **syslog_facility**. To record logs using syslog, add the following information to syslog daemon's configuration file:

```
local0.*  /var/log/omm
```

Default value: stderr

logging_collector

Parameter description: Specifies whether to enable the logger process to collect logs. This process captures log messages sent to **stderr** or **csvlog** and redirects them into log files.

This method is more effective than recording logs to syslog because some types of messages cannot be displayed in syslog output, such as messages indicating the loading failures of dynamic link libraries and error messages generated by scripts (for example, **archive_command**).

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

It is possible to log to **stderr** without using the logging collector and the log messages will go to where the server's **stderr** is directed. However, this method is only suitable for low log volumes due to difficulties in rotating log files.

Value range: Boolean

- **on** indicates that the log collection is enabled.
- **off** indicates that the log collection is disabled.

Default value: **on**

log_directory

Parameter description: Specifies the directory for storing log files when **logging_collector** is set to **on**. The value can be an absolute path, or relative to the data directory. The **log_directory** parameter can be dynamically modified using the **gs_guc reload** command. Only users with the **sysadmin** permission can access this parameter.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

- If this parameter is set to an invalid path, the database cannot be started.
- If you modify the **log_directory** parameter using the **gs_guc reload** command, and the specified path is valid, the log files are output to this new path. If the specified path is invalid, the log files are output to the valid path set last time and the database operation is not affected. The invalid value is still written into the configuration file.
- In the sandbox environment, the path cannot contain /var/chroot. For example, if the absolute path of log is **/var/chroot/var/lib/log/Ruby/pg_log/cn_log**, you only need to set the path to **/var/lib/log/Ruby/pg_log/cn_log**.

NOTE

- Valid path: Users have read and write permissions on the path.
- Invalid path: Users do not have read or write permissions on an invalid path.

Value range: a string

Default value: specified during installation

log_filename

Parameter description: Specifies the names of generated log files when **logging_collector** is set to **on**. The value is treated as a strftime pattern, so %-escapes can be used to specify time-varying file names. Only users with the **sysadmin** permission can access this parameter.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

- You are advised to use %-escapes to specify the log file names for efficient management of log files.
- If **log_destination** is set to **csvlog**, log files are output in CSV format with timestamped names, for example, **server_log.1093827753.csv**.

Value range: a string

Default value: **postgresql-***%Y-%m-%d_%H%M%S***.log**

log_file_mode

Parameter description: Specifies the permissions of log files when **logging_collector** is set to **on**. The parameter value is usually a number in the format acceptable to the **chmod** and **umask** system calls.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

- Before setting this parameter, set **log_directory** to store the logs to a directory other than the data directory.
- Do not make the log files world-readable because they might contain sensitive data.

Value range: an octal integer ranging from 0000 to 0777 (that is, 0 to 511 in the decimal format)

NOTE

- **0600** indicates that log files are readable and writable only to the server administrator.
- **0640** indicates that log files are readable and writable to members of the administrator's group.

Default value: 0600

log_truncate_on_rotation

Parameter description: Specifies the writing mode of the log files when **logging_collector** is set to **on**.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

A setting example is as follows:

Assume that you want logs to be kept for 7 days, a log file generated each day to be named **server_log.Mon** on Monday, **server_log.Tue** on Tuesday, and so forth, and this week's log files to be overwritten by next week's log files. Then you can set **log_filename** to **server_log.%a**, **log_truncate_on_rotation** to **on**, and **log_rotation_age** to **1440** (indicating that the valid duration of the log file is 24 hours).

Value range: Boolean

- **on** indicates that GaussDB overwrites the existing log files of the same name on the server.
- **off** indicates that GaussDB appends the logging messages to the existing log files of the same name on the server.

Default value: **off**

log_rotation_age

Parameter description: Specifies the interval for creating a log file when **logging_collector** is set to **on**. If the duration from the time when the last log file was created to the current time is greater than the value of **log_rotation_age**, a new log file will be generated.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 35791394. The unit is min. **0** indicates that the time-based creation of new log files is disabled.

Default value: **1440**

log_rotation_size

Parameter description: Specifies the maximum size of a server log file when **logging_collector** is set to **on**. If the total size of messages in a log file exceeds the specified value, a log file will be generated.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to *INT_MAX*/1024. The unit is KB.

0 indicates that the capacity-based creation of new log files is disabled.

It is recommended that the unit of the value be MB or bigger, so that log files can be of proper size.

Default value: 20 MB

syslog_facility

Parameter description: Specifies the syslog facility to be used when **log_destination** is set to **syslog**.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: enumerated values. Valid values are **local0**, **local1**, **local2**, **local3**, **local4**, **local5**, **local6**, and **local7**.

Default value: local0

syslog_ident

Parameter description: Specifies the identifier of GaussDB messages in syslog logs when **log_destination** is set to **syslog**.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string

Default value: **postgres**

event_source

Parameter description: This parameter takes effect only in a Windows environment and is not supported in GaussDB. It specifies the identifier of GaussDB messages in logs when **log_destination** is set to **eventlog**.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string

Default value: **PostgreSQL**

17.8.2 Logging Time

client_min_messages

Parameter description: Specifies which level of messages will be sent to the client. Each level covers all the levels following it. The lower the level is, the fewer messages are sent.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

A same value for **client_min_messages** and **log_min_messages** does not indicate the same level.

Value range: enumerated values. Valid values are **debug**, **debug5**, **debug4**, **debug3**, **debug2**, **debug1**, **info**, **log**, **notice**, **warning**, **error**, **fatal**, and **panic**. For details about the parameter, see **Table 17-2**. If the configured level is higher than **error**, for example, **fatal** or **panic**, the system changes the level to **error** by default.

Default value: **notice**

log_min_messages

Parameter description: Specifies which level of messages will be written into server logs. Each level covers all the levels following it. The lower the level is, the fewer messages will be written into the log.

This parameter is a SUSET parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

A same value for **client_min_messages** and **log_min_messages** does not indicate the same level. For some log information, after this parameter is enabled, you also need to set **logging_module** to enable log printing for the corresponding module.

Value range: enumerated values. Valid values are **debug**, **debug5**, **debug4**, **debug3**, **debug2**, **debug1**, **info**, **log**, **notice**, **warning**, **error**, **fatal**, and **panic**. For details about the parameter, see **Table 17-2**.

Default value: **warning**

log_min_error_statement

Parameter description: Controls which SQL statements that cause an error condition are recorded in the server log.

This parameter is a SUSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: enumerated values. Valid values are **debug**, **debug5**, **debug4**, **debug3**, **debug2**, **debug1**, **info**, **log**, **notice**, **warning**, **error**, **fatal**, and **panic**. For details about the parameter, see **Table 17-2**.

NOTE

- The default is **error**, indicating that statements causing errors, log messages, fatal errors, or panics will be logged.
- **panic**: This feature is disabled.

Default value: **error**

log_min_duration_statement

Parameter description: Specifies the threshold for logging the duration of a completed statement. If a statement runs for a period greater than or equal to the specified value, its duration will be logged.

Setting this parameter can be helpful in tracking down unoptimized queries. For clients using extended query protocols, the time required for parsing, binding, and executing steps are logged independently.

This parameter is a SUSET parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

When using this option together with **log_statement**, the text of statements that are logged because of **log_statement** will not be repeated in the duration log message. If you are not using **syslog**, it is recommended that you log the process ID (PID) or session ID using **log_line_prefix** so that you can link the statement message to the later duration message.

Value range: an integer ranging from –1 to 2147483647. The unit is ms.

- If this parameter is set to **250**, all SQL statements that run for 250 ms or longer will be logged.
- **0** indicates that the execution durations of all the statements are logged.
- **–1**: This function is disabled.

Default value: 3s (that is, 3000ms)

backtrace_min_messages

Parameter description: Prints the function's stack information to the server's log file if the information generated is greater than or equal to the level specified by this parameter.

This parameter is a SUSET parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

This parameter is used to locate problems on-site. Frequent stack printing will affect the system's overhead and stability. Therefore, set the value of this parameter to a rank other than **fatal** or **panic** during problem location.

Value range: enumerated values

Valid values include **debug**, **debug5**, **debug4**, **debug3**, **debug2**, **debug1**, **info**, **log**, **notice**, **warning**, **error**, **fatal**, and **panic**. For details about the parameter, see **Table 17-2**.

Default value: **panic**

Table 17-2 explains message severities used by GaussDB. If logging output is sent to **syslog** or **eventlog**, the severities are translated as shown in the table. (Note that the translation takes effect only in a Windows environment where GaussDB does not involve this parameter.)

Table 17-2 Message severity levels

Severity	Description	System Log	Event Log
debug[1-5]	Provides detailed debug information.	DEBUG	INFORMATIO N
log	Provides information of interest to administrators, for example, checkpoint activity.	INFO	INFORMATIO N
info	Provides information implicitly requested by users, for example, output from **VACUUM VERBOSE**.	INFO	INFORMATIO N
notice	Provides information that might be helpful to users, for example, truncation of long identifiers and index created as part of the primary key.	NOTICE	INFORMATIO N
warning	Provides warnings of likely problems, for example, **COMMIT** outside a transaction block.	NOTICE	WARNING
error	Provides an error that causes a command to terminate.	WARNING	ERROR
fatal	Provides the reason that causes a session to terminate.	ERR	ERROR

Severity	Description	System Log	Event Log
panic	Provides an error that caused all database sessions to terminate.	CRIT	ERROR

plog_merge_age

Parameter description: Specifies the output period of performance log data. The current feature is a lab feature. Contact Huawei technical support before using it.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

The parameter value is in milliseconds. You are advised to set it to a multiple of 1000. That is, the value is in seconds. The performance log files controlled by this parameter are stored in the *$GAUSSLOG*/**gs_profile/**<*node_name*> directory in .prf format. You are not advised to use this parameter externally.

Value range: a number ranging from 0 to 2147483647. The unit is ms.

0 indicates that the current session will not log performance data. A value other than 0 indicates that the current session will log performance data based on the period specified by this parameter.

A small value indicates that much data is logged, which seriously affects performance.

Default value: **0**

17.8.3 Logging Content

debug_print_parse

Parameter description: Specifies whether to print parsing tree results.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the printing is enabled.
- **off** indicates that the printing is disabled.

Default value: **off**

debug_print_rewritten

Parameter description: Specifies whether to print query rewriting results.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the printing is enabled.
- **off** indicates that the printing is disabled.

Default value: **off**

debug_print_plan

Parameter description: Specifies whether to print the query execution plan to logs.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the printing is enabled.
- **off** indicates that the printing is disabled.

Default value: **off**

NOTICE

- Debugging information about **debug_print_parse**, **debug_print_rewritten**, and **debug_print_plan** are printed only when the log level is set to **log** or higher. When these parameters are set to **on**, their debugging information will be recorded in server logs and will not be sent to client logs. You can change the log level by setting **client_min_messages** and **log_min_messages**.
- Do not invoke the **gs_encrypt_aes128** and **gs_decrypt_aes128** functions when **debug_print_plan** is set to **on**, preventing the risk of sensitive information disclosure. You are advised to filter parameter information of the **gs_encrypt_aes128** and **gs_decrypt_aes128** functions in the log files generated when **debug_print_plan** is set to **on** before providing the log files to external maintenance engineers for fault locating. After you finish using the logs, delete them as soon as possible.

debug_pretty_print

Parameter description: Indents the logs produced by **debug_print_parse**, **debug_print_rewritten**, and **debug_print_plan**. The output format is more readable but much longer than that generated when this parameter is set to **off**.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the indentation is enabled.
- **off** indicates that the indentation is disabled.

Default value: **on**

log_checkpoints

Parameter description: Specifies whether the statistics on checkpoints and restart points are recorded in the server logs. When this parameter is set to **on**, statistics on checkpoints and restart points are recorded in the log messages, including the number of buffers written and the time spent in writing them.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the statistics on checkpoints and restart points are recorded in the server logs.
- **off** indicates that the statistics on checkpoints and restart points are not recorded in the server logs

Default value: **off**

log_connections

Parameter description: Specifies whether to record connection request information of the client.

This parameter is a BACKEND parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

Some client programs, such as gsql, attempt to connect twice while determining if a password is required. In this case, duplicate "connection receive" messages do not necessarily indicate a problem.

Value range: Boolean

- **on** indicates that the request information is recorded.
- **off** indicates that the request information is not recorded.

Default value: **off**

log_disconnections

Parameter description: Specifies whether to record disconnection request information of the client.

This parameter is a BACKEND parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the request information is recorded.
- **off** indicates that the request information is not recorded.

Default value: **off**

log_duration

Parameter description: Specifies whether to record the duration of every completed SQL statement. For clients using extended query protocols, the time required for parsing, binding, and executing steps are logged independently.

This parameter is a **SUSET** parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **off**: Compared with this option, **log_min_duration_statement** forcibly records the query text.
- If this parameter is set to **on** and **log_min_duration_statement** is set to a positive value, the duration of each completed statement is logged but the query text is included only for statements exceeding the threshold. This behavior can be used for gathering statistics in high-load situation.

Default value: **off**

log_error_verbosity

Parameter description: Specifies the amount of detail written in the server log for each message that is logged.

This parameter is a SUSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: enumerated values

- **terse** indicates that the output excludes the DETAIL, HINT, QUERY, and CONTEXT error information.
- **verbose** indicates that the output includes the SQLSTATE error code, the source code file name, function name, and number of the line in which the error occurs.
- **default** indicates that the output includes the DETAIL, HINT, QUERY, and CONTEXT error information, and excludes the SQLSTATE error code, the source code file name, function name, and number of the line in which the error occurs.

Default value: **default**

log_hostname

Parameter description: By default, connection log messages only show the IP address of the connecting host. The host name can be recorded when this parameter is set to **on**. It may take some time to parse the host name. Therefore, the database performance may be affected.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the host name is simultaneously recorded.
- **off** indicates that the host name is not simultaneously recorded.

Default value: **off**

log_line_prefix

Parameter description: Specifies the prefix format of each log information. A prefix is a printf-style string that is output at the beginning of each line of the log. The "escape sequences" which begin with **%** are replaced with status information as listed in **Table 17-3**.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Table 17-3 Escape characters

Escape Character	Effect
%a	Application name
%u	Username
%d	Database name
%r	Remote host name or IP address and remote port. If **log_hostname** is set to **off**, only the IP address and remote port are displayed.
%h	Remote host name or IP address. If **log_hostname** is set to **off**, only the IP address is displayed.
%p	Thread ID
%t	Time stamp without milliseconds
%m	Timestamp with milliseconds
%n	Node from which an error is reported
%i	Command tag: type of command executed in the current session
%e	SQLSTATE error code
%c	Session ID: For details, see the note below the table.
%l	Number of the log line for each session or thread, starting at 1
%s	Process startup time
%v	Virtual transaction ID (backendID/ localXID)
%x	Transaction ID (**0** indicates that no transaction ID is assigned)

Escape Character	Effect
%q	Produces no output. If the current thread is a backend thread, this escape sequence is ignored and subsequent escape sequences are processed. Otherwise, this escape sequence and subsequent escape sequences are all ignored.
%S	Session ID
%T	Trace ID
%%	The character %

NOTE

The %c escape character prints a session ID consisting of two 4-byte hexadecimal numbers separated by a period (.). The numbers are the process startup time and the process ID. Therefore, %c can also be used as a space saving way of printing those items. For example, run the following query to generate the session ID from **pg_stat_activity**:

```
SELECT to_hex(EXTRACT(EPOCH FROM backend_start)::integer) || '.' ||
       to_hex(pid)
FROM pg_stat_activity;
```

- If you set a nonempty value for **log_line_prefix**, you should usually make its last character be a space, to provide visual separation from the rest of the log line. A punctuation character can be used, too.
- Syslog generates its own timestamp and process ID information. Therefore, you do not need to include those escapes characters when you are logging in to syslog.

Value range: a string

Default value: %m %n %u %d %h %p %S %x %a

NOTE

%m %n %u %d %h %p %S %x %a indicates the session start timestamp, error reporting node, username, database name, remote host name or IP address, thread ID, session ID, transaction ID, and application name.

log_lock_waits

Parameter description: If the time for which a session waits to acquire a lock is longer than the value of **deadlock_timeout**, this parameter specifies whether to record this message in the database. This is useful in determining if lock waits are causing poor performance.

This parameter is a SUSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the information is recorded.
- **off** indicates that the information is not recorded.

Default value: **off**

log_statement

Parameter description: Specifies which SQL statements are recorded. For clients using extended query protocols, logging occurs when an Execute message is received, and values of the Bind parameters are included (with any embedded single quotation marks doubled).

This parameter is a SUSET parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

Statements that contain simple syntax errors are not logged even if **log_statement** is set to **all**, because the log message is emitted only after basic parsing has been completed to determine the statement type. If an extended query protocol is used, statements that fail before the execution phase (during parse analysis or planning) are not logged, either. Set **log_min_error_statement** to **ERROR** or lower to log such statements.

Value range: enumerated values

- **none** indicates that no statement is recorded.
- **ddl** indicates that all data definition statements, such as CREATE, ALTER, and DROP, are recorded.
- **mod** indicates that all DDL statements and data modification statements, such as INSERT, UPDATE, DELETE, TRUNCATE, and COPY FROM, are recorded.
- **all** indicates that all statements, including the PREPARE, EXECUTE, and EXPLAIN ANALYZE statements, are recorded.

Default value: **none**

log_temp_files

Parameter description: Specifies whether to record the deletion information of temporary files. Temporary files can be created for sorting, hashing, and storing temporary querying results. If the recording is enabled, a log entry is generated for each temporary file when it is deleted.

This parameter is a SUSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from –1 to 2147483647. The unit is KB.

- A positive value indicates that the deletion information of temporary files whose size is larger than the specified value of **log_temp_files** is recorded.
- **0** indicates that the delete information of all temporary files is recorded.
- **–1** indicates that the delete information of any temporary files is not recorded.

Default value: **–1**

log_timezone

Parameter description: Specifies the time zone used for timestamps written in the server log. Different from **TimeZone**, this parameter takes effect for all sessions in the database.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string. You can obtain it by querying the **PG_TIMEZONE_NAMES** view.

Default value: Set this parameter based on the OS time zone.

NOTE

The default value will be changed when **gs_initdb** is used to set system environments.

logging_module

Parameter description: Specifies whether module logs are output on the server. This parameter is a session-level parameter, and you are advised not to use the gs_guc tool to set it.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string

Default value: **off**. All the module logs are not output on the server. You can view the logs by running **SHOW logging_module**.

```
ALL,on(),off(COMMAND,GUC,GSCLEAN,SLRU,MEM_CTL,AUTOVAC,CACHE,ADIO,SSL,GDS,TBLSPC,WLM,OBS,I
NDEX,EXECUTOR,OPFUSION,GPC,GSC,VEC_EXECUTOR,STREAM,LL
VM,OPT,OPT_REWRITE,OPT_JOIN,OPT_AGG,OPT_CHOICE,OPT_SUBPLAN,OPT_SETOP,OPT_SKEW,OPT_PLANN
ER,UDF,COOP_ANALYZE,WLMCP,ACCELERATE,MOT,PLANHINT,PARQUET,PGSTAT,
SNAPSHOT,XACT,HANDLE,CLOG,EC,REMOTE,CN_RETRY,PLSQL,TEXTSEARCH,SEQ,REDO,FUNCTION,PARSER,I
NSTR,WDR_SNAPSHOT,INCRE_CKPT,INCRE_BG_WRITER,DBL_WRT,RTO_RPO,
HEARTBEAT,COMM_IPC,COMM_PARAM,TIMESERIES,SCHEMA,SEGMENT_PAGE,LIGHTPROXY,HOTKEY,THREA
D_POOL,OPT_AI,WALRECEIVER,USTORE,UNDO,TIMECAPSULE,GEN_COL,DCF,DB4AI,PLDEBUGGER,ADVIS
OR,SEC,SEC_FE,SEC_LEGER,SEC_POLICY,SEC_SDD,SEC_TDE,COMM_PROXY,COMM_POOLER,VACUUM,JOB,SPI,
NEST_COMPILE,RESOWNER,LOGICAL_DECODE,GPRC,DISASTER_READ,REPSYNC,ENCODING_CHECK)
```

Setting method: Run **show logging_module** to view which modules are controllable. For example, the query output result is as follows:

```
openGauss=# show logging_module;
logging_module
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
----------------------------------------------
ALL,on(),off(COMMAND,GUC,GSCLEAN,SLRU,MEM_CTL,AUTOVAC,CACHE,ADIO,SSL,GDS,TBLSPC,WLM,OBS,I
NDEX,EXECUTOR,OPFUSION,GPC,GSC,VEC_EXECUTOR,STREAM,LL
VM,OPT,OPT_REWRITE,OPT_JOIN,OPT_AGG,OPT_CHOICE,OPT_SUBPLAN,OPT_SETOP,OPT_SKEW,OPT_PLANN
ER,UDF,COOP_ANALYZE,WLMCP,ACCELERATE,MOT,PLANHINT,PARQUET,PGSTAT,
SNAPSHOT,XACT,HANDLE,CLOG,EC,REMOTE,CN_RETRY,PLSQL,TEXTSEARCH,SEQ,REDO,FUNCTION,PARSER,I
NSTR,WDR_SNAPSHOT,INCRE_CKPT,INCRE_BG_WRITER,DBL_WRT,RTO_RPO,
HEARTBEAT,COMM_IPC,COMM_PARAM,TIMESERIES,SCHEMA,SEGMENT_PAGE,LIGHTPROXY,HOTKEY,THREA
D_POOL,OPT_AI,WALRECEIVER,USTORE,UNDO,TIMECAPSULE,GEN_COL,DCF,DB4AI,PLDEBUGGER,ADVIS
OR,SEC,SEC_FE,SEC_LEGER,SEC_POLICY,SEC_SDD,SEC_TDE,COMM_PROXY,COMM_POOLER,VACUUM,JOB,SPI,
NEST_COMPILE,RESOWNER,LOGICAL_DECODE,GPRC,DISASTER_READ,REPSYNC,ENCODING_CHECK)
(1 row)
```

Controllable modules are identified by uppercase letters, and the special ID **ALL** is used for setting all module logs. You can control the output of module logs by setting **logging_module** to **on** or **off**. Enable log output for SSL:

```
openGauss=# set logging_module='on(SSL)';
SET
openGauss=# show
logging_module;
       logging_module
-----------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------
ALL,on(SSL),off(COMMAND,GUC,GSCLEAN,SLRU,MEM_CTL,AUTOVAC,CACHE,ADIO,GDS,TBLSPC,WLM,OBS,I
NDEX,EXECUTOR,OPFUSION,GPC,GSC,VEC_EXECUTOR,STREAM,LLV
M,OPT,OPT_REWRITE,OPT_JOIN,OPT_AGG,OPT_CHOICE,OPT_SUBPLAN,OPT_SETOP,OPT_SKEW,OPT_PLANNE
R,UDF,COOP_ANALYZE,WLMCP,ACCELERATE,MOT,PLANHINT,PARQUET,PGSTAT,
SNAPSHOT,XACT,HANDLE,CLOG,EC,REMOTE,CN_RETRY,PLSQL,TEXTSEARCH,SEQ,REDO,FUNCTION,PARSER,I
NSTR,WDR_SNAPSHOT,INCRE_CKPT,INCRE_BG_WRITER,DBL_WRT,RTO_RPO,H
EARTBEAT,COMM_IPC,COMM_PARAM,TIMESERIES,SCHEMA,SEGMENT_PAGE,LIGHTPROXY,HOTKEY,THREAD_
POOL,OPT_AI,WALRECEIVER,USTORE,UNDO,TIMECAPSULE,GEN_COL,DCF,DB4AI,PLDEBUGGER,ADVISO
R,SEC,SEC_FE,SEC_LEGER,SEC_POLICY,SEC_SDD,SEC_TDE,COMM_PROXY,COMM_POOLER,VACUUM,JOB,SPI,N
EST_COMPILE,RESOWNER,LOGICAL_DECODE,GPRC,DISASTER_READ,REPSYNC,ENCODING_CHECK)
(1 row)
```

SSL log output is enabled.

The **ALL** identifier can be used to quickly enable or disable log output for all modules.

```
openGauss=# set logging_module='off(ALL)';
SET
openGauss=# show
logging_module;
         logging_module
-----------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------
ALL,on(),off(COMMAND,GUC,GSCLEAN,SLRU,MEM_CTL,AUTOVAC,CACHE,ADIO,SSL,GDS,TBLSPC,WLM,OBS,I
NDEX,EXECUTOR,OPFUSION,GPC,GSC,VEC_EXECUTOR,STREAM,LL
VM,OPT,OPT_REWRITE,OPT_JOIN,OPT_AGG,OPT_CHOICE,OPT_SUBPLAN,OPT_SETOP,OPT_SKEW,OPT_PLANN
ER,UDF,COOP_ANALYZE,WLMCP,ACCELERATE,MOT,PLANHINT,PARQUET,PGSTAT,
SNAPSHOT,XACT,HANDLE,CLOG,EC,REMOTE,CN_RETRY,PLSQL,TEXTSEARCH,SEQ,REDO,FUNCTION,PARSER,I
NSTR,WDR_SNAPSHOT,INCRE_CKPT,INCRE_BG_WRITER,DBL_WRT,RTO_RPO,
HEARTBEAT,COMM_IPC,COMM_PARAM,TIMESERIES,SCHEMA,SEGMENT_PAGE,LIGHTPROXY,HOTKEY,THREA
D_POOL,OPT_AI,WALRECEIVER,USTORE,UNDO,TIMECAPSULE,GEN_COL,DCF,DB4AI,PLDEBUGGER,ADVIS
OR,SEC,SEC_FE,SEC_LEGER,SEC_POLICY,SEC_SDD,SEC_TDE,COMM_PROXY,COMM_POOLER,VACUUM,JOB,SPI,
NEST_COMPILE,RESOWNER,LOGICAL_DECODE,GPRC,DISASTER_READ,REPSYNC,ENCODING_CHECK)
(1 row)

openGauss=# set logging_module='on(ALL)';
SET
openGauss=# show
logging_module;
        logging_module
-----------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------
ALL,on(COMMAND,GUC,GSCLEAN,SLRU,MEM_CTL,AUTOVAC,CACHE,ADIO,SSL,GDS,TBLSPC,WLM,OBS,INDE
X,EXECUTOR,OPFUSION,GPC,GSC,VEC_EXECUTOR,STREAM,LLVM,OPT
,OPT_REWRITE,OPT_JOIN,OPT_AGG,OPT_CHOICE,OPT_SUBPLAN,OPT_SETOP,OPT_SKEW,OPT_PLANNER,UDF,
COOP_ANALYZE,WLMCP,ACCELERATE,MOT,PLANHINT,PARQUET,PGSTAT,
SNAPSHOT,XACT,HANDLE,CLOG,EC,REMOTE,CN_RETRY,PLSQL,TEXTSEARCH,SEQ,REDO,FUNCTION,PARSER,I
NSTR,WDR_SNAPSHOT,INCRE_CKPT,INCRE_BG_WRITER,DBL_WRT,RTO_RPO,HEARTB
EAT,COMM_IPC,COMM_PARAM,TIMESERIES,SCHEMA,SEGMENT_PAGE,LIGHTPROXY,HOTKEY,THREAD_POOL,
OPT_AI,WALRECEIVER,USTORE,UNDO,TIMECAPSULE,GEN_COL,DCF,DB4AI,PLDEBUGGER,ADVISOR,SEC
,SEC_FE,SEC_LEGER,SEC_POLICY,SEC_SDD,SEC_TDE,COMM_PROXY,COMM_POOLER,VACUUM,JOB,SPI,NEST_C
OMPILE,RESOWNER,LOGICAL_DECODE,GPRC,DISASTER_READ,REPSYNC,ENCODING_CHECK),off()
(1 row)
```

Dependency: The value of this parameter depends on the settings of **log_min_messages**.

opfusion_debug_mode

Parameter description: Checks whether simple queries are optimized for debugging. If this parameter is set to **log**, you can view the specific reasons why queries are not optimized in the database node execution plans.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: enumerated values

- **off** indicates that reasons why queries are not optimized are not included.
- **log** indicates that reasons why queries are not optimized are included in the database node execution plan.

NOTICE

To view the reasons why queries are not optimized in the log, set **opfusion_debug_mode** to **log**, **log_min_messages** to **debug4**, and **logging_module** to **on(OPFUSION)**. Note that a large number of log messages may be generated. Therefore, execute only a small number of jobs during debugging.

Default value: **off**

enable_debug_vacuum

Parameter description: Specifies whether to allow output of some VACUUM-related logs for problem locating. This parameter is used only by developers. Common users are advised not to use it.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** or **true** indicates that generation of logs is allowed.
- **off** or **false** indicates that generation of logs is not allowed.

Default value: **off**

17.8.4 Using CSV Log Output

Prerequisites

- The **log_destination** parameter is set to **csvlog**.
- The **logging_collector** parameter is set to **on**.

Definition of csvlog

Log lines are emitted in comma separated values (CSV) format.

An example table definition for storing CSV-format log output is shown as follows:

```
CREATE TABLE gaussdb_log
(
log_time timestamp(3) with time zone,
node_name text,
user_name text,
database_name text,
process_id bigint,
connection_from text,
"session_id" text,
session_line_num bigint,
command_tag text,
session_start_time timestamp with time zone,
virtual_transaction_id text,
transaction_id bigint,
query_id bigint,
module text,
error_severity text,
sql_state_code text,
message text,
detail text,
hint text,
internal_query text,
internal_query_pos integer,
context text,
query text,
query_pos integer,
location text,
application_name text
);
```

For details, see **Table 17-4**.

Table 17-4 Meaning of each csvlog field

Column	Description	Column	Description
log_time	Timestamp in milliseconds	module	Module to which the log belongs
node_name	Node name	error_severity	ERRORSTATE code
user_name	Username	sql_state_code	SQLSTATE code
database_name	Database name	message	Error message
process_id	Process ID	detail	Detailed error message
connection_from	Port number of the client host	hint	Prompt message
session_id	Session ID	internal_query	Internal query (This field is used to query the information leading to errors if any.)

Column	Description	Column	Description
session_line_num	Number of lines in each session	internal_query_pos	Pointer for an internal query
command_tag	Command tag	context	Environment
session_start_time	Start time of a session	query	Character count at the position where errors occur
virtual_transaction_id	Regular transaction	query_pos	Pointer at the position where errors occur
transaction_id	Transaction ID	location	Position where errors occur in the GaussDB source code if **log_error_verbosity** is set to **verbose**
query_id	Query ID	application_name	Application name

Run the following command to import a log file to this table:

```
COPY gaussdb_log FROM '/opt/data/pg_log/logfile.csv' WITH csv;
```

NOTE

The log name (**logfile.csv**) here needs to be replaced with the name of a log generated.

Simplifying Input

Simplify importing CSV log files by performing the following operations:

- Set **log_filename** and **log_rotation_age** to provide a consistent, predictable naming solution for log files. By doing this, you can predict when an individual log file is complete and ready to be imported.
- Set **log_rotation_size** to **0** to disable size-based log rollback, as it makes the log file name difficult to predict.
- Set **log_truncate_on_rotation** to **on** so that old log data cannot be mixed with the new one in the same file.

17.9 Alarm Detection

During the running of the database, error scenarios can be detected so that users are informed of the errors in time. You can view the **system_alarm** log written by the alarm in the *$GAUSSLOG*/**cm** directory.

enable_alarm

Parameter description: Specifies whether to enable the alarm detection thread to detect fault scenarios that may occur in the database.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the alarm detection thread is enabled.
- **off** indicates that the alarm detection thread is disabled.

Default value: **on**

NOTE

This parameter takes effect only on DNs.

connection_alarm_rate

Parameter description: Specifies the ratio restriction on the maximum number of allowed parallel connections to the database. The maximum number of concurrent connections to the database is **max_connections** x **connection_alarm_rate**.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a floating point number ranging from 0.0 to 1.0

Default value: **0.9**

alarm_report_interval

Parameter description: specifies the interval at which an alarm is reported.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer. The unit is s.

Default value: 10

alarm_component

Parameter description: Certain alarms are suppressed during alarm reporting. That is, the same alarm will not be repeatedly reported by an instance within the period specified by **alarm_report_interval**. Its default value is **10s**. In this case, the parameter specifies the location of the alarm component that is used to process alarm information. Only the sysadmin user can access this parameter.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string

- If **--alarm-type** in the **gs_preinstall** script is set to **5**, no third-party component is connected and alarms are written into the **system_alarm** log.

In this case, the value of **alarm_component** is **/opt/huawei/snas/bin/snas_cm_cmd**.

- If **--alarm-type** in the **gs_preinstall** script is set to **1**, a third-party component is connected. In this case, the value of **alarm_component** is the absolute path of the executable program of the third-party component.

Default value: **/opt/huawei/snas/bin/snas_cm_cmd**

table_skewness_warning_threshold

Parameter description: Specifies the threshold for triggering a table skew alarm.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a floating point number ranging from 0 to 1

Default value: **1**

table_skewness_warning_rows

Parameter description: Specifies the number of rows for triggering a table skew alarm.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to *INT_MAX*

Default value: **100000**

17.10 Statistics During the Database Running

17.10.1 Query and Index Statistics Collector

The query and index statistics collector is used to collect statistics during database running. The statistics include the times of inserting and updating a table and index, the number of disk blocks and tuples, and the time required for the last cleanup and analysis on each table. The statistics can be viewed by querying system view families pg_stats and pg_statistic. The following parameters are used to set the statistics collection feature in the server scope.

track_activities

Parameter description: Collects statistics about the commands that are being executed in session. For a stored procedure, if this parameter is enabled, you can view the PERFORM statement, stored procedure calling statement, SQL statement, and OPEN CURSOR statement that are being executed in the stored procedure in the pg_stat_activity view.

This parameter is a SUSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the statistics collection function is enabled.
- **off** indicates that the statistics collection function is disabled.

Default value: **on**

track_counts

Parameter description: Collects statistics about database activities.

This parameter is a SUSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the statistics collection function is enabled.
- **off** indicates that the statistics collection function is disabled.

NOTE

Database statistics are required when the autovacuum process checks for databases that need to be vacuumed. Therefore, the default value is set to **on**.

Default value: **on**

track_io_timing

Parameter description: Collects statistics about I/O timing in the database. The I/O timing statistics can be queried by using the **pg_stat_database** parameter.

This parameter is a SUSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- If this parameter is set to **on**, the collection function is enabled. In this case, the collector repeatedly queries the operating system at the current time. As a result, large number of costs may occur on some platforms. Therefore, the default value is set to **off**.
- **off** indicates that the statistics collection function is disabled.

Default value: **off**

track_functions

Parameter description: Collects statistics of the number and duration of function invocations.

This parameter is a SUSET parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

When the SQL functions are set to inline functions queried by the invoking, these SQL functions cannot be traced no matter these functions are set or not.

Value range: enumerated values

- **pl** indicates that only procedural language functions are traced.
- **all** indicates that SQL language functions area traced.
- **none** indicates that the function tracing function is disabled.

Default value: **none**

track_activity_query_size

Parameter description: Specifies byte counts of the current running commands used to trace each active session.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 100 to 102400

Default value: 1024

update_process_title

Parameter description: Collects statistics updated with a process name each time the server receives a new SQL statement.

The process name can be viewed by running the **ps** command.

This parameter is an INTERNAL parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the statistics collection function is enabled.
- **off** indicates that the statistics collection function is disabled.

Default value: **off**

stats_temp_directory

Parameter description: Specifies the directory for storing temporary statistics. Only the sysadmin user can access this parameter.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

If a RAM-based file system directory is used, the actual I/O cost can be lowered and the performance can be improved.

Value range: a string

Default value: **pg_stat_tmp**

track_thread_wait_status_interval

Parameter description: Specifies the interval of collecting the thread status information.

This parameter is a SUSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: 0 to 1 day. The unit is min.

Default value: **30min**

enable_save_datachanged_timestamp

Parameter description: Specifies whether to record the time when **INSERT**, **UPDATE**, **DELETE**, or **EXCHANGE**/**TRUNCATE**/**DROP PARTITION** is performed on table data.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the time when an operation is performed on table data will be recorded.
- **off** indicates that the time when an operation is performed on table data will not be recorded.

Default value: **on**

track_sql_count

Parameter description: Collects statistics about the statements (**SELECT**, **INSERT**, **UPDATE**, **MERGE INTO**, and **DELETE**) that are being executed in a session.

In the x86-based centralized deployment scenario, the hardware configuration specifications are 32-core CPU and 256 GB memory. When the Benchmark SQL 5.0 tool is used to test performance, the performance fluctuates by about 0.8% by enabling or disabling this parameter.

This parameter is a SUSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the statistics collection function is enabled.
- **off** indicates that the auditing function is disabled.

Default value: **on**

NOTE

- The **track_sql_count** parameter is restricted by the **track_activities** parameter.
  - If **track_activities** is set to **on** and **track_sql_count** is set to **off**, a warning message indicating that **track_sql_count** is disabled will be displayed in logs when the **gs_sql_count** view is queried.
  - If both **track_activities** and **track_sql_count** are set to **off**, two warning messages indicating that **track_activities** is disabled and **track_sql_count** is disabled will be displayed in logs when the views are queried.
  - If **track_activities** is set to **off** and **track_sql_count** is set to **on**, a warning message indicating that **track_activities** is disabled will be displayed in logs when the views are queried.
- If **track_sql_count** is set to **off**, querying the **gs_sql_count** or **pgxc_sql_count** view returns **0**.

17.10.2 Performance Statistics

During the running of the database, the lock access, disk I/O operation, and invalid message processing are involved. All these operations are the bottleneck of the database performance. The performance statistics provided by GaussDB can facilitate the performance fault location.

Generating Performance Statistics Logs

Parameter description: For each query, the following four parameters record the performance statistics of corresponding modules in the server log:

- The **og_parser_stats** parameter records the performance statistics of a parser in the server log.
- The **log_planner_stats** parameter records the performance statistics of a query optimizer in the server log.
- The **log_executor_stats** parameter records the performance statistics of an executor in the server log.
- The **log_statement_stats** parameter records the performance statistics of the whole statement in the server log.

All these parameters can only provide assistant analysis for administrators, which are similar to the getrusage() of the Linux OS.

These parameters are SUSET parameters. Set them based on instructions provided in Table 10-1.

NOTICE

- The **log_statement_stats** records the total statement statistics whereas other parameters record statistics only about their corresponding modules.
- The **log_statement_stats** parameter cannot be enabled together with any parameter recording statistics about a module.

Value range: Boolean

- **on** indicates that performance statistics are recorded.

- **off** indicates that performance statistics are not recorded.

Default value: **off**

17.11 Workload Management

The current feature is a lab feature. Contact Huawei technical support before using it.

If database resource usage is not controlled, concurrent tasks may preempt resources. As a result, the OS will be overloaded and cannot respond to user tasks; or even crash and cannot provide any services to users. The GaussDB workload management balances the database workload based on available resources to prevent database overloads.

use_workload_manager

Parameter description: Specifies whether to enable the resource management function.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the resource management function is enabled.
- **off** indicates that the resource management function is disabled.

NOTE

- If method 2 in **Table 10-1** is used to change the parameter value, the new value takes effect only for the threads that are started after the change. In addition, the new value does not take effect for new jobs that are executed by backend threads and reused threads. You can make the new value take effect for these threads by using **kill session** or restarting the node.
- After the value of **use_workload_manager** changes from **off** to **on**, statistics about storage resources when **use_workload_manager** was **off** are not collected. To collect statistics about such resources, run the following statement:

```
select gs_wlm_readjust_user_space(0);
```

Default value: **on**

cgroup_name

Parameter description: Specifies the name of the cgroup in use or changes the priority of items in the queue of the cgroup.

If you set **cgroup_name** and then **session_respool**, the cgroup associated with **session_respool** take effect. If you reverse the order, the cgroup associated with **cgroup_name** take effect.

If the Workload cgroup level is specified during the **cgroup_name** change, the database does not check the cgroup level. The level ranges from 1 to 10.

This parameter is a USERSET parameter. Set it based on method 3 in **Table 10-1**.

You are advised not to set **cgroup_name** and **session_respool** at the same time.

Value range: a string

Default value: **InvalidGroup**

cpu_collect_timer

Parameter description: Specifies how frequently CPU data is collected during statement execution on database nodes.

The database administrator should set a proper collection frequency based on system resources (for example, CPU, I/O, and memory resources). A too small value will affect the execution efficiency and a too large value will reduce the accuracy of exception handling.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 1 to *INT_MAX*. The unit is s.

Default value: **30**

memory_tracking_mode

Parameter description: Specifies the memory information recording mode.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range:

- **none** indicates that memory statistics are not collected.
- **peak** indicates that statistics on the peak value of the query level memory are collected. The value is recorded in the database and can also be output by using **explain analyze**.
- **normal** indicates that memory statistics are collected in real time but no file is generated.
- **executor** indicates that a statistics file is generated, containing the context information of all allocated memory used on the execution layer.
- **fullexec** indicates that a statistics file is generated, containing the information about all memory contexts requested by the execution layer.

Default value: **none**

memory_detail_tracking

Parameter description: Specifies the memory context allocation priority of a thread and the plannodeid of the query for which the current thread is running.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string

Default value: empty

NOTICE

You are advised to retain the default value for this parameter.

enable_resource_track

Parameter description: Specifies whether the real-time resource monitoring function is enabled.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the resource monitoring is enabled.
- **off** indicates that the resource monitoring is disabled.

Default value: **on**

enable_resource_record

Parameter description: Specifies whether resource monitoring records are archived.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the resource monitoring records are archived.
- **off** indicates that the resource monitoring records are not archived.

Default value: **off**

enable_logical_io_statistics

Parameter description: Specifies whether to enable the logical I/O statistics function during resource monitoring. If this function is enabled, fields in the PG_TOTAL_USER_RESOURCE_INFO view such as **read_kbytes**, **write_kbytes**, **read_counts**, **write_counts**, **read_speed**, and **write_speed** collect statistics on the number of logical read/write bytes, number of read/write times, and the read/ write speed.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the function is enabled.
- **off** indicates that the function is disabled.

Default value: **on**

enable_user_metric_persistent

Parameter description: Specifies whether the user historical resource monitoring dumping function is enabled. If this parameter is set to **on**, data in the

PG_TOTAL_USER_RESOURCE_INFO view is periodically sampled and saved to the system catalog **GS_WLM_USER_RESOURCE_HISTORY**.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

on indicates that the historical monitoring data of user resources is dumped.

off indicates that the historical monitoring data of user resources is not dumped

Default value: **on**

user_metric_retention_time

Parameter description: Specifies the retention period of the user historical resource monitoring data. This parameter is valid only when **enable_user_metric_persistent** is set to **on**.

This parameter is a SIGHUP parameter. Set it based on methods 1 and 2 provided in **Table 10-2**.

Value range: an integer ranging from 0 to 730. The unit is day.

If this parameter is set to **0**, the historical monitoring data of user resources is permanently stored.

If the value is greater than **0**, the historical monitoring data of user resources is stored for the specified number of days.

Default value: **7**

enable_instance_metric_persistent

Parameter description: Specifies whether the instance resource monitoring data is dumped. When this parameter is set to **on**, the instance monitoring data is saved to the system catalog **GS_WLM_INSTANCE_HISTORY**.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the instance resource monitoring data is dumped.
- **off** indicates that the instance resource monitoring data is not dumped.

Default value: **on**

instance_metric_retention_time

Parameter description: Specifies the retention days of the historical monitoring data of instance resources. This parameter is valid only when **enable_instance_metric_persistent** is set to **on**.

This parameter is a USERSET parameter. Set it based on methods 1 and 2 provided in **Table 10-2**.

Value range: an integer ranging from 0 to 3650. The unit is day.

- If this parameter is set to **0**, the historical monitoring data of instance resources is permanently stored.
- If the value is greater than **0**, the historical monitoring data of instance resources is stored for the specified number of days.

Default value: 7

resource_track_level

Parameter description: Specifies the resource monitoring level of the current session. This parameter is valid only when **enable_resource_track** is set to **on**.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Valid value: enumerated values

- **none** indicates that resources are not monitored.
- **query** indicates that resources used at the query level are monitored.
- **operator** indicates that resources used at query and operator levels are monitored.

Default value: **query**

resource_track_cost

Parameter description: Specifies the minimum execution cost for resource monitoring on statements in the current session. This parameter is valid only when **enable_resource_track** is set to **on**.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from –1 to *INT_MAX*

- **–1** indicates that resource monitoring is disabled.
- A value ranging from 0 to 9 indicates that statements whose execution cost is greater than or equal to 10 will be monitored.
- A value greater than or equal to **10** indicates that statements whose execution cost exceeds this value will be monitored.

Default value: **100000**

resource_track_duration

Parameter description: Specifies the minimum statement execution time that determines whether information about jobs of a statement recorded in the real-time view will be dumped to a historical view after the statement is executed. Job information will be dumped from the real-time view (with the suffix **statistics**) to a historical view (with the suffix **history**) if the statement execution time is no less than this value. This parameter is valid only when **enable_resource_track** is set to **on**.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to *INT_MAX*. The unit is s.

- **0** indicates that historical information about all statements recorded in the real-time resource monitoring view are archived.
- If the value is greater than **0**, historical information about a statement whose execution time exceeds this value will be archived.

Default value: **1min**

disable_memory_protect

Parameter description: Stops memory protection. To query system views when system memory is insufficient, set this parameter to **on** to stop memory protection. This parameter is used only to diagnose and debug the system when system memory is insufficient. Set it to **off** in other scenarios.

This parameter is a USERSET parameter and is valid only for the current session. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that memory protection stops.
- **off** indicates that memory is protected.

Default value: **off**

query_band

Parameter description: Specifies the job type of the current session.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string

Default value: empty

memory_fault_percent

Parameter description: Specifies the percentage of memory application failures during the memory fault test. This parameter is used only in the DEBUG version.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 2147483647

Default value: **0**

enable_bbox_dump

Parameter description: Specifies whether the black box function is enabled. The core files can be generated even when the core dump mechanism is not configured in the system.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the black box function is enabled.
- **off** indicates that the black box function is disabled.

Default value: **on**

NOTICE

The generation of core files by the black box depends on the open **ptrace** API of the operating system. If the permission is insufficient (**errno = 1**), ensure that **/proc/sys/kernel/yama/ptrace_scope** is configured properly.

bbox_dump_count

Parameter description: Specifies the maximum number of core files that are generated by GaussDB and can be stored in the path specified by **bbox_dump_path**. If the number of core files exceeds this value, old core files will be deleted. This parameter is valid only when **enable_bbox_dump** is set to **on**

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 1 to 20

Default value: **8**

NOTE

When core files are generated during concurrent SQL statement execution, the number of files may be larger than the value of **bbox_dump_count**.

bbox_dump_path

Parameter description: Specifies the path where the black box core files are generated. This parameter is valid only when **enable_bbox_dump** is set to **on**.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string

Default value: empty The default path where the black box core files are generated is **/proc/sys/kernel/core_pattern**. If the path is not a directory or you do not have the write permission for the directory, the black box core files will be generated in the data directory of the database. Alternatively, use the directory specified during installation.

enable_ffic_log

Parameter description: Specifies whether to enable the first failure information capture (FFIC) function.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the FFIC function is enabled.
- **off** indicates that the FFIC function is disabled.

Default value: **on**

io_limits

Parameter description: Specifies the upper limit of Input/output operations per second (IOPS).

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 1073741823

Default value: **0**

io_priority

Parameter description: Specifies the I/O priority for jobs that consume many I/O resources. It takes effect when the I/O usage reaches 50%.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Valid value: enumerated values

- **None** indicates no control.
- **Low** indicates that the IOPS is reduced to 10% of the original value.
- **Medium** indicates that the IOPS is reduced to 20% of the original value.
- **High** indicates that the IOPS is reduced to 50% of the original value.

Default value: none

io_control_unit

Parameter description: Specifies the unit used to count the number of I/Os during I/O control in row-store scenarios.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Set a certain number of I/Os as one unit. This unit is used during the I/O control.

Value range: an integer ranging from 1000 to 1000000

Default value: **6000**

session_respool

Parameter description: Specifies the resource pool associated with the current session.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

If you set **cgroup_name** and then **session_respool**, the cgroup associated with **session_respool** take effect. If you reverse the order, the cgroup associated with **cgroup_name** take effect.

If the Workload cgroup level is specified during the **cgroup_name** change, the database does not check the cgroup level. The level ranges from 1 to 10.

You are advised not to set **cgroup_name** and **session_respool** at the same time.

Value range: a string. This parameter can be set to the resource pool configured through **create resource pool**.

Default value: **invalid_pool**

session_statistics_memory

Parameter description: Specifies the memory size of a real-time query view.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 5 x 1024 to 50% of **max_process_memory**. The unit is KB.

Default value: **5MB**

topsql_retention_time

Parameter description: Specifies the retention period of historical TopSQL data in the **gs_wlm_operator_info** table.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 730. The unit is day.

- If it is set to **0**, the data is stored permanently.
- If the value is greater than **0**, the data is stored for the specified number of days.

Default value: **0**

session_history_memory

Parameter description: Specifies the memory size of a historical query view.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 10 x 1024 to 50% of **max_process_memory**. The unit is KB.

Default value: 10 MB

transaction_pending_time

Parameter description: maximum queuing time of transaction block statements and stored procedure statements.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from –1 to *INT_MAX/2*. The unit is s.

- **–1** or **0** indicates that no queuing timeout is specified for transaction block statements and stored procedure statements. The statements can be executed when resources are available.
- A value greater than **0** indicates that if transaction block statements and stored procedure statements have been queued for a time longer than the specified value, they are forcibly executed regardless of the current resource situation.

Default value: **0**

bbox_blanklist_items

Parameter description: Specifies the anonymized data items of black box core files. This parameter is valid only when **enable_bbox_dump** is set to **on**.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string of sensitive data items separated by commas (,).

Default value: empty which indicates that all supported sensitive data items of the core files generated by the black box are anonymized.

Currently, the following data items can be anonymized:

- SHARED_BUFFER: data buffer
- XLOG_BUFFER: redo log buffer
- DW_BUFFER: doublewrite data buffer
- XLOG_MESSAGE_SEND: buffer for sending primary/standby replication logs
- WALRECIVER_CTL_BLOCK: buffer for receiving primary/standby replication logs
- DATA_MESSAGE_SEND: buffer for sending primary/standby replication data
- DATA_WRITER_QUEUE: buffer for receiving primary/standby replication data

current_logic_cluster

Parameter description: Displays the name of the current logical database instance.

This parameter is a fixed INTERNAL parameter and cannot be modified.

Value range: a string

Default value: empty

17.12 Automatic Vacuuming

The **autovacuum** process automatically runs the **VACUUM** and **ANALYZE** statements to recycle the record space marked as deleted and update statistics about the table.

autovacuum

Parameter description: Specifies whether to start the **autovacuum** process in the database. Ensure that the **track_counts** parameter is set to **on** before starting the automatic cleanup process.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-2**.

NOTE

- Set the **autovacuum** parameter to **on** if you want to start the automatic cleanup of abnormal two-phase transactions when the system recovers from faults.
- If **autovacuum** is set to **on** and **autovacuum_max_workers** to **0**, the autovacuum process is started only when the system recovers from faults to clean up abnormal two-phase transactions.
- If **autovacuum** is set to **on** and the value of **autovacuum_max_workers** is greater than **0**, the system will automatically clean up the two-phase transactions and processes after recovering from faults.

NOTICE

Even if **autovacuum** is set to **off**, the autovacuum process will be started automatically when a transaction ID wraparound is about to occur. When a **CREATE DATABASE** or **DROP DATABASE** operation fails, it is possible that the transaction has been committed or rolled back on some nodes whereas some nodes are still in the prepared state. In this case, perform the following operations to manually restore the nodes:

Procedure

1. Use the gs_clean tool (setting the **option** parameter to -N) to query the xid of the abnormal two-phase transaction and nodes in the prepared status.
2. Log in to the nodes in the prepared state. Administrators connect to an available database and run the **set xc_maintenance_mode = on** statement.
3. Commit or roll back the two-phase transaction based on the global transaction status.

Value range: Boolean

- **on** indicates that the **autovacuum** process is started.
- **off** indicates that the **autovacuum** process is not started.

Default value: **on**

autovacuum_mode

Parameter description: Specifies whether the autoanalyze or autovacuum function is started. This parameter is valid only when **autovacuum** is set to **on**.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-2**.

Value range: enumerated values

- **analyze** indicates that only autoanalyze is performed.
- **vacuum** indicates that only autovacuum is performed.
- **mix** indicates that both autoanalyze and autovacuum are performed.
- **none** indicates that neither of them is performed.

Default value: **mix**

autoanalyze_timeout

Parameter description: Specifies the timeout period of autoanalyze. If the duration of autoanalyze on a table exceeds the value of **autoanalyze_timeout**, the autoanalyze is automatically canceled.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-2**.

Value range: an integer ranging from 0 to 2147483. The unit is s.

Default value: **5min** (300s)

autovacuum_io_limits

Parameter description: Specifies the upper limit of I/Os triggered by the autovacuum process per second.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-2**.

Value range: an integer. The value can be **–1** or a number ranging from 0 to 1073741823. **–1** indicates that the default cgroup is used.

Default value: **–1**

log_autovacuum_min_duration

Parameter description: Records each step performed by the autovacuum process to the server log when the execution time of the autovacuum process is greater than or equal to a certain value. This parameter helps track the autovacuum behavior.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-2**.

A setting example is as follows:

Set the **log_autovacuum_min_duration** parameter to 250 ms to record the actions of autovacuum if it runs for 250 ms or longer.

Value range: an integer ranging from –1 to 2147483647. The unit is ms.

- **0** indicates that all autovacuum actions are recorded in the log.
- **–1** indicates that all autovacuum actions are not recorded in the log.
- A value other than **–1** indicates that a message is recorded when an autovacuum action is skipped due to a lock conflict.

Default value: **–1**

autovacuum_max_workers

Parameter description: Specifies the maximum number of autovacuum worker threads that can run at the same time. The upper limit of this parameter is related to the values of **max_connections** and **job_queue_processes**.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-2**.

Value range: an integer. The minimum value is **0**, indicating that autovacuum is not enabled. The theoretical maximum value is **262143**, but the actual maximum value is a dynamic value calculated by the following formula: 262143 - **max_connections** - **job_queue_processes** - Number of auxiliary threads - Number of autovacuum launcher threads - 1. The number of auxiliary threads and the number of autovacuum launcher threads are specified by two macros. Their default values are **20** and **2** respectively.

Default value: **3**

autovacuum_naptime

Parameter description: Specifies the interval between activity rounds for the autovacuum process.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-2**.

Value range: an integer ranging from 1 to 2147483. The unit is s.

Default value: **10min** (600s)

autovacuum_vacuum_threshold

Parameter description: Specifies the threshold for triggering the **VACUUM** operation. When the number of deleted or updated records in a table exceeds the specified threshold, the **VACUUM** operation is executed on this table.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-2**.

Value range: an integer ranging from 0 to 2147483647.

Default value: **50**

autovacuum_analyze_threshold

Parameter description: Specifies the threshold for triggering the **ANALYZE** operation. When the number of deleted, inserted, or updated records in a table exceeds the specified threshold, the **ANALYZE** operation is executed on this table.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-2**.

Value range: an integer ranging from 0 to 2147483647.

Default value: **50**

autovacuum_vacuum_scale_factor

Parameter description: Specifies a fraction of the table size added to the **autovacuum_vacuum_threshold** parameter when deciding whether to vacuum a table.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-2**.

Value range: a floating point number ranging from 0.0 to 100.0

Default value: **0.2**

autovacuum_analyze_scale_factor

Parameter description: Specifies a fraction of the table size added to the **autovacuum_analyze_threshold** parameter when deciding whether to analyze a table.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-2**.

Value range: a floating point number ranging from 0.0 to 100.0

Default value: **0.1**

autovacuum_freeze_max_age

Parameter description: Specifies the maximum age (in transactions) that a table's **pg_class.relfrozenxid** field can attain before a VACUUM operation is performed.

- The old files under the subdirectory of **pg_clog/** can also be deleted by the **VACUUM** operation.
- Even if the **autovacuum** process is not started, the system will invoke the process to prevent transaction ID wraparound.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-2**.

Value range: an integer ranging from 100000 to 576460752303423487

Default value: **4000000000**

autovacuum_vacuum_cost_delay

Parameter description: Specifies the value of the cost delay used in the **autovacuum** operation.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-2**.

Value range: an integer ranging from –1 to 100. The unit is ms. **–1** indicates that the normal vacuum cost delay is used.

Default value: **20ms**

autovacuum_vacuum_cost_limit

Parameter description: sets the value of the cost limit used in the **autovacuum** operation.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-2**.

Value range: an integer ranging from –1 to 10000 **–1** indicates that the normal vacuum cost limit is used.

Default value: **–1**

defer_csn_cleanup_time

Parameter description: Specifies the interval of recycling transaction IDs.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-2**.

Value range: an integer ranging from 0 to *INT_MAX*. The unit is ms.

Default value: **5s** (5000 ms)

17.13 Default Settings of Client Connection

17.13.1 Statement Behavior

This section describes related default parameters involved in the execution of SQL statements.

search_path

Parameter description: Specifies the order in which schemas are searched when an object is referenced with no schema specified. The value of this parameter consists of one or more schema names. Different schema names are separated by commas (,).

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

- If the schema of temporary tables exists in the current session, the schema can be listed in **search_path** by using the alias **pg_temp**, for example, **'pg_temp,public'**. The schema of temporary tables has the highest search priority and is always searched before all the other schemas specified in **pg_catalog** and **search_path**. Therefore, do not explicitly specify **pg_temp** to be searched after other schemas in **search_path**. This setting will not take effect and an error message will be displayed. If the alias **pg_temp** is used, the temporary schema will be searched only for tables, views, and data types, and not for functions or operators.
- The system catalog schema, **pg_catalog**, has the second highest search priority and is the first to be searched among all the schemas, excluding **pg_temp**, specified in **search_path**. Therefore, do not explicitly specify **pg_catalog** to be searched after other schemas in **search_path**. This setting will not take effect and an error message will be displayed.

- When an object is created without a specific target schema, the object will be placed in the first valid schema listed in **search_path**. An error is reported if the search path is empty.
- The current effective value of the search path can be examined through the SQL function **current_schema**. This is different from examining the value of **search_path**, because the **current_schema** function displays the first valid schema name in **search_path**.

Value range: a string

NOTE

- When this parameter is set to **"$user", public**, shared use of a database (where no users have private schemas, and all share use of public), private per-user schemas and combinations of them are supported. Other effects can be obtained by modifying the default search path setting, either globally or per-user.
- When this parameter is set to a null string (''), the system automatically converts it into a pair of double quotation marks ("").
- If the content contains double quotation marks, the system considers them as insecure characters and converts each double quotation mark into a pair of double quotation marks.

Default value: **"$user",public**

NOTE

$user indicates the name of the schema with the same name as the current session user. If the schema does not exist, **$user** will be ignored.

current_schema

Parameter description: Specifies the current schema.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string

Default value: **"$user",public**

NOTE

$user indicates the name of the schema with the same name as the current session user. If the schema does not exist, **$user** will be ignored.

default_tablespace

Parameter description: Specifies the default tablespace of the created objects (tables and indexes) when a **CREATE** command does not explicitly specify a tablespace.

- The value of this parameter is either the name of a tablespace, or an empty string that indicates the use of the default tablespace of the current database. If a non-default tablespace is specified, users must have CREATE privilege for it. Otherwise, creation attempts will fail.
- This parameter is not used for temporary tables. For them, the **temp_tablespaces** is used instead.

- This parameter is not used when users create databases. By default, a new database inherits its tablespace setting from the template database.

  This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string. An empty string indicates that the default tablespace is used.

Default value: empty

temp_tablespaces

Parameter description: Specifies one or more tablespaces to which temporary objects (temporary tables and their indexes) will be created when a CREATE command does not explicitly specify a tablespace. Temporary files for sorting large data sets are created in these tablespaces.

The value of this parameter can be a list of names of tablespaces. When there is more than one name in the list, GaussDB chooses a random tablespace from the list upon the creation of a temporary object each time. However, within a transaction, successively created temporary objects are placed in successive tablespaces in the list. If the element selected from the list is an empty string, GaussDB will automatically use the default tablespace of the current database instead.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string An empty string indicates that all temporary objects are created only in the default tablespace of the current database. For details, see **default_tablespace**.

Default value: empty

check_function_bodies

Parameter description: Specifies whether to enable validation of the function body string during the execution of **CREATE FUNCTION**. Verification is occasionally disabled to avoid problems, such as forward references when you restore function definitions from a dump. After the function is enabled, the word syntax of the PL/SQL in the stored procedure is verified, including the data type, statement, and expression. The SQL statements in the stored procedure are not checked in the Create phase. Instead, they are checked during running.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that validation of the function body string is enabled during the execution of **CREATE FUNCTION**.
- **off** indicates that validation of the function body string is disabled during the execution of **CREATE FUNCTION**.

Default value: **on**

default_transaction_isolation

Parameter description: Specifies the default isolation level of each transaction.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

NOTE

The current version does not support the setting of the default transaction isolation level. The default value is **read committed**. Do not change the value.

Value range: enumerated values

- **read committed** indicates that the data read by a transaction is committed at the moment it is read.
- **repeatable read** indicates that the data that has been read by the current transaction cannot be modified by other transactions until the current transaction completes, thereby preventing unrepeatable reads.
- **serializable**: Currently, this isolation level is not supported in GaussDB. It is equivalent to **repeatable read**.

Default value: read committed

default_transaction_read_only

Parameter description: Specifies whether each new transaction is in read-only state.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

CAUTION

If this parameter is set to **on**, the DML and write transactions cannot be executed.

Value range: Boolean

- **on** indicates that the transaction is in read-only state.
- **off** indicates that the transaction is in read/write state.

Default value: **off**

default_transaction_deferrable

Parameter description: Specifies the default deferrable status of each new transaction. It currently has no effect on read-only transactions or those running at isolation levels lower than serializable.

GaussDB does not support the serializable isolation level. Therefore, the parameter takes no effect.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that a transaction is delayed by default.
- **off** indicates that a transaction is not delayed by default.

Default value: **off**

session_replication_role

Parameter description: Specifies the behavior of replication-related triggers and rules for the current session.

This parameter is a SUSET parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

Setting this parameter will discard all the cached execution plans.

Value range: enumerated values

- **origin** indicates that the system copies operations such as insert, delete, and update from the current session.
- **replica** indicates that the system copies operations such as insert, delete, and update from other places to the current session.
- **local** indicates that the system will detect the role that has logged in to the database when using the function to copy operations and will perform related operations.

Default value: **origin**

statement_timeout

Parameter description: If the statement execution time (starting from the time the server receives the command) is longer than the duration specified by the parameter, error information is displayed and the statement exits.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**. The default value is **0**, indicating that the parameter does not take effect.

Value range: an integer ranging from 0 to 2147483647. The unit is ms.

Default value: **0**

vacuum_freeze_min_age

Parameter description: Specifies whether VACUUM replaces the **xmin** column of a record with **FrozenXID** when scanning a table (in the same transaction).

This parameter is a **USERSET** parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 576460752303423487

NOTE

Although you can set this parameter to any value, VACUUM will limit the effective value to half the value of **autovacuum_freeze_max_age** by default.

Default value: **2000000000**

vacuum_freeze_table_age

Parameter description: Specifies when VACUUM scans the whole table and freezes old tuples. VACUUM performs a full table scan if the difference between the current transaction ID and the value of pg_class.relfrozenxid64 is greater than the specified time.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 576460752303423487

NOTE

Although you can set this parameter to any value, **VACUUM** will limit the effective value to 95% of **autovacuum_freeze_max_age** by default. Therefore, a periodical manual **VACUUM** has a chance to run before an anti-wraparound autovacuum is launched for the table.

Default value: **4000000000**

bytea_output

Parameter description: Specifies the output format for values of the bytea type.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: enumerated values

- **hex** indicates that the binary data is converted to hexadecimal format.
- **escape** indicates that the traditional PostgreSQL format is used. It takes the approach of representing a binary string as a sequence of ASCII characters, while converting those bytes that cannot be represented as an ASCII character into special escape sequences.

Default value: **hex**

xmlbinary

Parameter description: Specifies how binary values are to be encoded in XML.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: enumerated values

- base64
- hex

Default value: **base64**

xmloption

Parameter description: Specifies whether DOCUMENT or CONTENT is implicit when converting between XML and string values.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: enumerated values

- **document** indicates an HTML document.
- **content** indicates a common string.

Default value: **content**

max_compile_functions

Parameter description: Specifies the maximum number of function compilation results stored in the server. Excessive functions and compilation results of stored procedures may occupy large memory space. Setting this parameter to a proper value can reduce the memory usage and improve system performance.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 1 to 2147483647

Default value: 1000

gin_pending_list_limit

Parameter description: Specifies the maximum size of the GIN pending list which is used when **fastupdate** is enabled. If the list grows larger than this maximum size, it is cleaned up by moving the entries in it to the main GIN data structure in batches. This setting can be overridden for individual GIN indexes by changing index storage parameters.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer. The minimum value is 64 and the maximum value is INT_MAX. The default unit is KB.

Default value: **4 MB**

17.13.2 Locale and Formatting

This section describes parameters related to the time format setting.

DateStyle

Parameter description: Specifies the display format for date and time values, as well as the rules for interpreting ambiguous date input values.

This variable contains two independent components: the output format specifications (ISO, Postgres, SQL, or German) and the input/output order of year/month/day (DMY, MDY, or YMD). The two components can be set separately or together. The keywords Euro and European are synonyms for DMY. The keywords US, NonEuro, and NonEuropean are synonyms for MDY.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string

Default value: **ISO, MDY**

NOTE

gs_initdb will initialize this parameter so that its value is the same as that of **lc_time**.

Setting Suggestions: The ISO format is recommended. Postgres, SQL, and German use abbreviations for time zones, such as **EST**, **WST**, and **CST**. These abbreviations can be ambiguous. For example, **CST** can represent Central Standard Time (USA) UT-6:00, Central Standard Time (Australia) UT+9:30, and China Standard Time UT+8:00. This may lead to incorrect time zone conversion and cause errors.

IntervalStyle

Parameter description: Specifies the display format for interval values.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: enumerated values

- **sql_standard** indicates that output matching SQL standards will be generated.
- **postgres** indicates that output matching PostgreSQL 8.4 will be generated when the **DateStyle** parameter is set to **ISO**.
- **postgres_verbose** indicates that output matching PostgreSQL 8.4 will be generated when the **DateStyle** parameter is set to **non_ISO**.
- **iso_8601** indicates that output matching the time interval "format with designators" defined in ISO 8601 will be generated.
- **a** indicates the output result that matches the numtodsinterval function. For details, see **numtodsinterval**.

NOTICE

The **IntervalStyle** parameter also affects the interpretation of ambiguous interval input.

Default value: **postgres**

TimeZone

Parameter description: Specifies the time zone for displaying and interpreting timestamps.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string. You can obtain it by querying the **PG_TIMEZONE_NAMES** view.

Default value: **PRC**

NOTE

gs_initdb will set a time zone value that is consistent with the system environment.

timezone_abbreviations

Parameter description: Specifies the time zone abbreviations that will be accepted by the server.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string. You can obtain it by querying the **pg_timezone_names** view.

Default value: **Default**

NOTE

Default indicates an abbreviation that works in most of the world, which is applicable to most cases. There are also other abbreviations, such as **Australia** and **India** that can be defined for a particular installation. For other time zone abbreviations, you need to set them in the corresponding configuration files before creating the database.

extra_float_digits

Parameter description: Adjusts the number of digits displayed for floating-point values, including float4, float8, and geometric data types. The parameter value is added to the standard number of digits (FLT_DIG or DBL_DIG as appropriate).

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from –15 to 3

NOTE

- This parameter can be set to **3** to include partially-significant digits. It is especially useful for dumping float data that needs to be restored exactly.
- This parameter can also be set to a negative value to suppress unwanted digits.

Default value: **0**

client_encoding

Parameter description: Specifies the client-side encoding (character set).

Set this parameter based on the situation of the front-end services. Try to keep the encoding consistent on the client and server to improve efficiency.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: encoding compatible with PostgreSQL. **UTF8** indicates that the database encoding is used.

NOTE

- You can run the **locale -a** command to check the system-supported locales and the corresponding encodings, and select one as required.
- By default, **gs_initdb** will initialize the setting of this parameter based on the current system environment. You can also run the **locale** command to check the current configuration environment.
- To use consistent encoding for communication within the database, you are advised to retain the default value of **client_encoding**. Modification to this parameter in the **postgresql.conf** file (by using the **gs_guc** tool, for example) does not take effect.

Default value: **UTF8**

Recommended value: **SQL_ASCII** or **UTF8**

lc_messages

Parameter description: Specifies the language in which messages are displayed.

- Acceptable values are system-related.
- On some systems, this locale category does not exist. Setting this variable will still work, but there will be no effect. In addition, translated messages for the desired language may not exist. In this case, you can still see the English messages.

  This parameter is a SUSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string

NOTE

- You can run the **locale -a** command to check the system-supported locales and the corresponding encodings, and select one as required.
- By default, **gs_initdb** will initialize the setting of this parameter based on the current system environment. You can also run the **locale** command to check the current configuration environment.

Default value: **C**

lc_monetary

Parameter description: Specifies the display format of monetary values. It affects the output of functions such as **to_char**. Acceptable values are system-related.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string

NOTE

- You can run the **locale -a** command to check the system-supported locales and the corresponding encodings, and select one as required.
- By default, **gs_initdb** will initialize the setting of this parameter based on the current system environment. You can also run the **locale** command to check the current configuration environment.

Default value: C

lc_numeric

Parameter description: Specifies the display format of numbers. It affects the output of functions such as **to_char**. Acceptable values are system-related.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string

NOTE

- You can run the **locale -a** command to check the system-supported locales and the corresponding encodings, and select one as required.
- By default, **gs_initdb** will initialize the setting of this parameter based on the current system environment. You can also run the **locale** command to check the current configuration environment.

Default value: C

lc_time

Parameter description: Specifies the display format of time and locale. It affects the output of functions such as **to_char**. Acceptable values are system-related.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string

NOTE

- You can run the **locale -a** command to check the system-supported locales and the corresponding encodings, and select one as required.
- By default, **gs_initdb** will initialize the setting of this parameter based on the current system environment. You can also run the **locale** command to check the current configuration environment.

Default value: C

default_text_search_config

Parameter description: Specifies the text search configuration.

If the specified text search configuration does not exist, an error will be reported. If the specified text search configuration is deleted, set **default_text_search_config** again. Otherwise, an error will be reported, indicating incorrect configuration.

- The text search configuration is used by text search functions that do not have an explicit argument specifying the configuration.
- When a configuration file matching the environment is determined, **gs_initdb** will initialize the configuration file with a setting that corresponds to the environment.

  This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string

NOTE

GaussDB supports the following two configurations: **pg_catalog.english** and **pg_catalog.simple**.

Default value: **pg_catalog.english**

17.13.3 Other Default Parameters

This section describes the default database loading parameters.

dynamic_library_path

Parameter description: Specifies the path that the system will search for a shared database file that is dynamically loadable. When a dynamically loadable module needs to be opened and the file name specified in the **CREATE FUNCTION** or **LOAD** command does not have a directory component, the system will search this path for the required file. Only the sysadmin user can access this parameter.

The value of **dynamic_library_path** must be a list of absolute paths separated by colons (:). When the name of a path starts with the special variable $libdir, the variable will be replaced with the directory in which the module provided by the GaussDB is installed. For example:

```
dynamic_library_path = '/usr/local/lib/gaussdb:/opt/testgs/lib:$libdir'
```

This parameter is a SUSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string

NOTE

If the value of this parameter is set to an empty character string, the automatic path search is turned off.

Default value: **$libdir**

gin_fuzzy_search_limit

Parameter description: Specifies the upper limit of the size of the set returned by GIN indexes.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 2147483647

Default value: **0**

local_preload_libraries

Parameter description: Specifies one or more shared libraries that are to be preloaded at connection start. If multiple libraries are to be loaded, separate their names with commas (,). All library names are converted to lower case unless double-quoted.

- Any user can change this option. Therefore, library files that can be loaded are restricted to those saved in the **plugins** subdirectory of the standard library installation directory. It is the database administrator's responsibility to ensure that libraries in this directory are all safe. Entries in **local_preload_libraries** can specify the library directory explicitly, for example, **$libdir/plugins/mylib**, or just specify the library name, for example, **mylib**. (**mylib** is equivalent to **$libdir/plugins/mylib**.)
- Unlike **shared_preload_libraries**, there are no differences in performance between loading a module at session start or doing this during the session. The intent of this feature is to allow debugging or performance-measurement libraries to be loaded into specific sessions without an explicit LOAD command. For example, debugging can be enabled under a given user name by setting this parameter to **ALTER USER SET**.
- If a specified library is not found, the connection attempt will fail.
- Every GaussDB-supported library has a "magic block" that is checked to guarantee compatibility. For this reason, non-GaussDB-supported libraries cannot be loaded in this way.

This parameter is a BACKEND parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string

Default value: **empty**

17.14 Lock Management

In GaussDB, a deadlock may occur when concurrently executed transactions compete for resources. This section describes parameters used for managing transaction locks.

deadlock_timeout

Parameter description: Specifies the time, in milliseconds, to wait on a lock before checking whether there is a deadlock condition. When the applied lock exceeds the preset value, the system will check whether a deadlock occurs. This parameter takes effect only for common locks.

- The check for deadlock is relatively expensive. Therefore, the server does not check it when waiting for a lock every time. Deadlocks do not frequently occur when the system is running. Therefore, the system just needs to wait on the lock for a while before checking for a deadlock. Increasing this value reduces the time wasted in needless deadlock checks, but slows down reporting of real deadlock errors. On a heavily loaded server, you may need to raise it. The value you have set needs to exceed the transaction time. By doing this, the possibility that a lock will be checked for deadlocks before it is released will be reduced.

- When **log_lock_waits** is set to **on**, **deadlock_timeout** determines a waiting time to write the lock waiting time information during query execution to logs. To study the lock delay, you can set **deadlock_timeout** to a value smaller than the normal value.

This parameter is a SUSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 1 to 2147483647. The unit is ms.

Default value: **1s**

lockwait_timeout

Parameter description: Specifies the timeout for attempts to acquire a lock. If the time spent in waiting for a lock exceeds the specified time, an error is reported. This parameter takes effect only for common locks.

This parameter is a SUSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to *INT_MAX*. The unit is ms.

Default value: **20min**

update_lockwait_timeout

Parameter description: Specifies the maximum duration that a lock waits for concurrent updates on a row to complete when the concurrent update feature is enabled. If the time spent in waiting for a lock exceeds the specified time, an error is reported. This parameter takes effect only for common locks.

This parameter is a SUSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 2147483647. The unit is ms.

Default value: **2min** (120000 ms)

max_locks_per_transaction

Parameter description: Determines the average number of object locks allocated for each transaction.

- The size of the shared lock table is calculated under the condition that a maximum of *N* independent objects need to be locked at any time. *N* = **max_locks_per_transaction** x (**max_connections** + **max_prepared_transactions**). Objects whose amount does not exceed the preset number can be locked simultaneously at any time. You may need to increase this value if many different tables are modified in a single transaction. This parameter can only be set at database start.
- Increasing the value of this parameter may cause GaussDB to request more System V-shared memory than the OS's default configuration allows.
- When running a standby server, you must set this parameter to a value that is no less than that on the primary server. Otherwise, queries will not be allowed on the standby server.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 10 to *INT_MAX*

Default value: **256**

max_pred_locks_per_transaction

Parameter description: Specifies the average number of predicate locks allocated for each transaction.

- The size of the shared predicate lock table is calculated under the condition that a maximum of *N* independent objects need to be locked at any time. *N* = **max_pred_locks_per_transaction** x (**max_connections** + **max_prepared_transactions**). Objects whose amount does not exceed the preset number can be locked simultaneously at any time. You may need to increase this value if many different tables are modified in a single transaction. This parameter can only be set at server start.
- Increasing the value of this parameter may cause GaussDB to request more System V-shared memory than the OS's default configuration allows.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 10 to *INT_MAX*

Default value: **64**

gs_clean_timeout

Parameter description: Specifies the average interval for clearing temporary tables on the primary node.

- When the database connection is terminated abnormally, temporary tables may exist. In this case, you need to call the **gs_clean** tool to clear the temporary tables in the database.
- If this parameter is set to a larger value, the time for clearing GaussDB temporary tables may be prolonged.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 2147483. The unit is s.

Default value: **1min**

partition_lock_upgrade_timeout

Parameter description: Specifies the timeout for attempts to upgrade an exclusive lock (read allowed) to an access exclusive lock (read/write blocked) on a partitioned table during the execution of some query statements. If there are concurrent read transactions running, the lock upgrade will need to wait. This parameter sets the waiting timeout for lock upgrade attempts.

- When you do **MERGE PARTITION** and **CLUSTER PARTITION** on a partitioned table, temporary tables are used for data rearrangement and file exchange. To concurrently perform as many operations as possible on the partitions, exclusive locks are acquired for the partitions during data rearrangement and access exclusive locks are acquired during file exchange.

- Generally, a partition waits until it acquires a lock, or a timeout occurs if the partition waits for a period longer than the value specified by the **lockwait_timeout** parameter.
- When doing **MERGE PARTITION** or **CLUSTER PARTITION** on a partitioned table, an access exclusive lock needs to be acquired during file exchange. If the lock fails to be acquired, the acquisition is retried at an interval of 50 ms until timeout occurs. The **partition_lock_upgrade_timeout** parameter specifies the time to wait before the lock acquisition attempt times out.
- If this parameter is set to **–1**, the lock upgrade never times out. The lock upgrade is continuously retried until it succeeds.

  This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from –1 to 3000. The unit is s.

Default value: **1800**

fault_mon_timeout

Parameter description: Specifies the period for detecting lightweight deadlocks. This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 1440. The unit is minute.

Default value: **5min**

enable_online_ddl_waitlock

Parameter description: Specifies whether to block DDL operations to wait for the release of database locks, such as **pg_advisory_lock**. This parameter is mainly used in online OM operations and you are not advised to modify the settings.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that DDL operations will be blocked to wait for the lock release.
- **off** indicates that DDL operations will not be blocked.

Default value: **off**

xloginsert_locks

Parameter description: Specifies the number of locks on concurrent write-ahead logging. This parameter is used to improve the efficiency of writing write-ahead logs.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 1 to 1000

Default value: 16

num_internal_lock_partitions

Parameter description: Specifies the number of internal lightweight lock partitions. It is mainly used for performance optimization in various scenarios. The content is organized in the KV format of keywords and numbers. Different types of locks are separated by commas (,). The sequence does not affect the setting result. For example, **CLOG_PART=256,CSNLOG_PART=512** is equivalent to **CSNLOG_PART=512,CLOG_PART=256**. If you set the same keyword multiple times, only the latest setting takes effect. For example, if you set **CLOG_PART** to **256** and **CLOG_PART** to **2**, the value of **CLOG_PART** is **2**. If no keyword is set, the default value is used. The usage description, maximum value, minimum value, and default value of each lock type are as follows:

- **CLOG_PART**: number of Clog file controllers. Increasing the value of this parameter improves the Clog writing efficiency and transaction submission performance, but increases the memory usage. Decreasing the value of this parameter reduces the memory usage, but may increase the conflict of writing Clogs and affect the performance. The value ranges from 1 to 256.
- **CSNLOG_PART**: number of CSNLOG file controllers. Increasing the value of this parameter improves the CSNLOG log writing efficiency and transaction submission performance, but increases the memory usage. Decreasing the value of this parameter reduces the memory usage, but may increase the conflict of writing CSNLOG logs and affect the performance. The value ranges from 1 to 512.
- **LOG2_LOCKTABLE_PART**: two logarithms of the number of common table lock partitions. Increasing the value can improve the concurrency of obtaining locks in the normal process, but may increase the time required for transferring and clearing locks. When waiting events occur in **LockMgrLock**, you can increase the value to improve the performance. The minimum value is 4, that is, the number of lock partitions is 16. The maximum value is 16, that is, the number of lock partitions is 65536.
- **TWOPHASE_PART**: number of partitions of the two-phase transaction lock. Increasing the value can increase the number of concurrent two-phase transaction commits. The value ranges from 1 to 64.
- **FASTPATH_PART**: maximum number of locks that each thread can obtain without using the primary lock table. Increasing the value of this parameter will consume more memory. The value ranges from 20 to 10000.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string

Default value:

- **CLOG_PART**: **256**
- **CSNLOG_PART**: **512**
- **LOG2_LOCKTABLE_PART**: **4**
- **TWOPHASE_PART**: **1**
- **FASTPATH_PART**: **20**

17.15 Version and Platform Compatibility

17.15.1 Compatibility with Earlier Versions

This section describes the parameters that control the backward compatibility and external compatibility of GaussDB. A backward compatible database supports applications of earlier versions. This section describes parameters used for controlling backward compatibility of a database.

array_nulls

Parameter description: Controls whether the array input parser recognizes unquoted NULL as a null array element.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that null values can be entered in arrays.
- **off** indicates backward compatibility with the old behavior. Arrays containing the value **NULL** can still be created when this parameter is set to **off**.

Default value: **on**

backslash_quote

Parameter description: Controls whether a single quotation mark can be represented by \' in a string text.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

When the string text meets the SQL standards, \ has no other meanings. This parameter only affects the handling of non-standard-conforming string texts, including escape string syntax (E'...').

Value range: enumerated values

- **on** indicates that the use of \' is always allowed.
- **off** indicates that the use of \' is rejected.
- **safe_encoding** indicates that the use of \' is allowed only when client encoding does not allow ASCII \ within a multibyte character.

Default value: **safe_encoding**

escape_string_warning

Parameter description: Specifies whether to issue a warning when a backslash (\) is used as an escape in an ordinary character string.

- Applications that wish to use a backslash (\) as an escape need to be modified to use escape string syntax (E'...'). This is because the default behavior of ordinary character strings treats the backslash as an ordinary character in each SQL standard.
- This variable can be enabled to help locate codes that need to be changed.
- If E'...' is used as an escape, logs may be incomplete in some scenarios.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

Default value: **on**

lo_compat_privileges

Parameter description: Specifies whether to enable backward compatibility for the privilege check of large objects.

This parameter is a SUSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

on indicates that the privilege check is disabled when users read or modify large objects. This setting is compatible with versions earlier than PostgreSQL 9.0.

Default value: **off**

quote_all_identifiers

Parameter description: Specifies whether to forcibly quote all identifiers even if they are not keywords when the database generates SQL. This will affect the output of **EXPLAIN** and the results of functions, such as pg_get_viewdef. For details, see the **--quote-all-identifiers** parameter of **gs_dump**.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the forcible quoting is enabled.
- **off** indicates that the forcible quoting is disabled.

Default value: **off**

sql_inheritance

Parameter description: Controls the inheritance semantics. This parameter specifies the access policy of descendant tables. **off** indicates that subtables cannot be accessed by commands. That is, the ONLY keyword is used by default. This setting is compatible with earlier versions.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that subtables can be accessed.
- **off** indicates that subtables cannot be accessed.

Default value: **on**

standard_conforming_strings

Parameter description: Controls whether ordinary string texts ('...') treat backslashes as ordinary texts as specified in the SQL standard.

- Applications can check this parameter to determine how string texts will be processed.
- It is recommended that characters be escaped by using the escape string syntax (E'...').

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that backslashes are treated as ordinary texts.
- **off** indicates that backslashes are not treated as ordinary texts.

Default value: **on**

synchronize_seqscans

Parameter description: Controls sequential scans of tables to synchronize with each other, so that concurrent scans read the same data block at about the same time and share the I/O workload.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that a scan may start in the middle of the table and then "wrap around" the end to cover all rows to synchronize with the activity of scans already in progress. This may result in unpredictable changes in the row ordering returned by queries that have no ORDER BY clause.
- **off** indicates that the scan always starts from the table heading.

Default value: **on**

enable_beta_features

Parameter description: Specifies whether to enable some features that are not officially released and are used only for POC verification. Exercise caution when enabling these extended features because they may cause errors in some scenarios.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the features are enabled for forward compatibility. Note that enabling them may cause errors in certain scenarios.
- **off** indicates that the features are disabled.

Default value: **off**

default_with_oids

Parameter description: Specifies whether **CREATE TABLE** and **CREATE TABLE AS** include an **OID** field in newly-created tables if neither **WITH OIDS** nor **WITHOUT OIDS** is specified. It also determines whether OIDs will be included in tables created by **SELECT INTO**.

It is not recommended that OIDs be used in user tables. Therefore, this parameter is set to **off** by default. When OIDs are required for a particular table, **WITH OIDS** needs to be specified during the table creation.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that **CREATE TABLE** and **CREATE TABLE AS** can include an **OID** field in newly-created tables.
- **off** indicates that **CREATE TABLE** and **CREATE TABLE AS** cannot include any OID field in newly-created tables.

Default value: **off**

17.15.2 Platform and Client Compatibility

Many platforms use the database system. External compatibility of the database system provides a lot of convenience for platforms.

convert_string_to_digit

Parameter description: Specifies the implicit conversion priority, which determines whether to preferentially convert strings into numbers.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that strings are preferentially converted into numbers.
- **off** indicates that strings are not preferentially converted into numbers.

Default value: **on**

NOTICE

Adjusting this parameter will change the internal data type conversion rule and cause unexpected behaviors. Exercise caution when performing this operation.

nls_timestamp_format

Parameter description: Specifies the default timestamp format.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string

Default value: **DD-Mon-YYYY HH:MI:SS.FF AM**

max_function_args

Parameter description: Specifies the maximum number of parameters allowed for a function.

This parameter is a fixed INTERNAL parameter and cannot be modified.

Value range: an integer

Default value: **8192**

transform_null_equals

Parameter description: Specifies whether expressions of the form expr = NULL (or NULL = expr) are treated as expr IS NULL. They return true if expr evaluates to the **NULL** value, and false otherwise.

- The correct SQL-standard-compliant behavior of **expr = NULL** is to always return **NULL** (unknown).
- Filtered forms in Microsoft Access generate queries that appear to use **expr = NULL** to test for null values. If you turn this option on, you can use this interface to access the database.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-2**.

Value range: Boolean

- **on** indicates that expressions of the form **expr = NULL** (or **NULL = expr**) are treated as expr **IS NULL**.
- **off** indicates that **expr = NULL** always returns **NULL** (unknown).

Default value: **off**

NOTE

New users are always confused about the semantics of expressions involving **NULL** values. Therefore, **off** is used as the default value.

support_extended_features

Parameter description: Specifies whether extended database features are supported.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-2**.

Value range: Boolean

- **on** indicates that extended database features are supported.
- **off** indicates that extended database features are not supported.

Default value: **off**

sql_compatibility

Parameter description: Specifies the type of mainstream database with which the SQL syntax and statement behavior of the database is compatible. This parameter is an INTERNAL parameter. It can be viewed but cannot be modified.

Value range: enumerated type

- **A** indicates that the database is compatible with the Oracle database.
- **B** indicates that the database is compatible with the MySQL database.
- **C** indicates that the database is compatible with the Teradata database.
- **PG** indicates that the database is compatible with the PostgreSQL database.

Default value: **A**

NOTICE

- This parameter can be set only when you run the **CREATE DATABASE** command to create a database.
- In the database, this parameter must be set to a specific value. It can be set to **A** or **B** and cannot be changed randomly. Otherwise, the setting is not consistent with the database behavior.

behavior_compat_options

Parameter description: Specifies database compatibility behavior. Multiple items are separated by commas (,).

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string

Default value: **""**

NOTE

- Currently, only compatibility configuration items in **Table 17-5** are supported.
- Multiple items are separated by commas (,), for example, **set behavior_compat_options='end_month_calculate,display_leading_zero';**.

Table 17-5 Compatibility configuration items

Configuration Item	Behavior
display_leading_zero	Specifies how floating point numbers are displayed. ● If this item is not specified, for a decimal number between –1 and 1, the 0 before the decimal point is not displayed. For example, 0.25 is displayed as **.25**. ● If this item is specified, for a decimal number between –1 and 1, the 0 before the decimal point is displayed. For example, 0.25 is displayed as **0.25**.

end_month_calculate	Specifies the calculation logic of the add_months function.
	Assume that the two parameters of the add_months function are **param1** and **param2**, and that the month of **param1** and **param2** is **result**.
	● If this item is not specified, and the **Day** of **param1** indicates the last day of a month shorter than **result**, the **Day** in the calculation result will equal that in **param1**. For example:
	`openGauss=# select add_months('2018-02-28',3) from sys_dummy;`
	`add_months`
	`---------------------`
	`2018-05-28 00:00:00`
	`(1 row)`
	● If this item is specified, and the **Day** of **param1** indicates the last day of a month shorter than **result**, the **Day** in the calculation result will equal that in **result**. For example:
	`openGauss=# select add_months('2018-02-28',3) from sys_dummy;`
	`add_months`
	`---------------------`
	`2018-05-31 00:00:00`
	`(1 row)`

compat_analyze_sample	Specifies the sampling behavior of the **ANALYZE** operation. If this item is specified, the sample collected by the ANALYZE operation will be limited to around 30,000 records, controlling database node memory consumption and maintaining the stability of ANALYZE.
bind_schema_tablespace	Binds a schema with the tablespace with the same name. If a tablespace name is the same as *sche_name*, **default_tablespace** will also be set to *sche_name* if **search_path** is set to *sche_name*.

Configuration Item	Behavior
bind_procedure_searchpath	Specifies the search path of the database object for which no schema name is specified. If no schema name is specified for a stored procedure, the search is performed in the schema to which the stored procedure belongs. If the stored procedure is not found, the following operations are performed: ● If this item is not specified, the system reports an error and exits. ● If this item is specified, the search continues based on the settings of **search_path**. If the issue persists, the system reports an error and exits.
correct_to_number	Specifies the compatibility of the to_number() result. If this item is specified, the result of the **to_number()** function is the same as that of PG11. Otherwise, the result is the same as that of the O database.
unbind_divide_bound	Specifies the range check on the result of integer division. If this item is specified, you do not need to check the range of the division result. For example, the result of INT_MIN/(-1) can be *INT_MAX*+1. If this item is not specified, an out-of-bounds error is reported because the result is greater than *INT_MAX*.
convert_string_digit_to_numeric	Determines whether to convert columns of the character string type to those of the numeric type before columns of these two types are compared.

return_null_string	Specifies how to display the empty result (empty string '') of the lpad() and rpad() functions.
	● If this item is not specified, the empty string is displayed as **NULL**.
	`openGauss=# select length(lpad('123',0,'*')) from sys_dummy;`
	`length`
	`--------`
	
	`(1 row)`
	● If this item is specified, the empty string is displayed as single quotation marks ('').
	`openGauss=# select length(lpad('123',0,'*')) from sys_dummy;`
	`length`
	`--------`
	`0`
	`(1 row)`

Configuration Item	Behavior
compat_concat_variadic	Specifies the compatibility of variadic results of the concat() and concat_ws() functions. If this item is specified and a concat function has a parameter of the variadic type, different result formats in A database and Teradata are retained. If this item is not specified and a concat function has a parameter of the variadic type, same result formats in A database and Teradata are retained, and the results are the same as those in A database. This option has no effect on MY because MY has no variadic type.
merge_update_multi	When MERGE INTO... WHEN MATCHED THEN UPDATE (see **MERGE INTO**) and INSERT... ON DUPLICATE KEY UPDATE (see **INSERT**) are used, control the UPDATE behavior if a piece of target data in the target table conflicts with multiple pieces of source data. If this item is specified and the preceding scenario exists, the system performs multiple UPDATE operations on the conflicting row. If this item is not specified and the preceding scenario exists, an error is reported, that is, the MERGE or INSERT operation fails.
plstmt_implicit_savepoint	Determines whether the execution of an UPDATE statement in a stored procedure has an independent subtransaction. If this parameter is set, the implicit savepoint is enabled before executing each UPDATE statement in the stored procedure, and the subtransaction is rolled backed to the latest savepoint in the EXCEPTION block by default, ensuring that only the modification of failed statements is rolled back. This option is used to be compatible with the **EXCEPTION** behavior of the O database.

hide_tailing_zero	Configuration item for numeric display. If this parameter is not set, numeric data is displayed in the specified precision. If this parameter is set, the trailing zero after the decimal point is hidden in all scenarios where numeric values are output, even the precision format is specified.
	`set behavior_compat_options='hide_tailing_zero';`
	`select cast(123.123 as numeric(15,10));`
	`numeric`
	`---------`
	`123.123`
	`(1 row)`

rownum_type_compat	Specifies the ROWNUM type. The default value is **BIGINT**. After this parameter is specified, the value is changed to **NUMERIC**.

Configuration Item	Behavior
aformat_null_test	Specifies the logic for checking whether **rowtype** is not null. When this parameter is set, if one column in a row is not empty, **true** is returned for checking whether **rowtype** is not null. When this parameter is not set, if all columns in a row are not empty, true is returned for checking whether **rowtype** is not null. This parameter has no influence on checking whether **rowtype** is not null.
aformat_regexp_match	Determines the matching behavior of regular expression functions. When this parameter is set and **sql_compatibility** is set to **A** or **B**, the options supported by the **flags** parameter of the regular expression are changed as follows: 1. . By default, the character '\n' cannot be matched. 2. When **flags** contains the **n** option, the character '\n' can be matched. 3. The **regexp_replace(source, pattern replacement)** function replaces all matching substrings. 4. **regexp_replace(source, pattern, replacement, flags)** returns null when the value of **flags** is **''** or null. Otherwise, the meanings of the options supported by the **flags** parameter of the regular expression are as follows: 1. . By default, the character '\n' can be matched. 2. The **n** option in **flags** indicates that the multi-line matching mode is used. 3. The **regexp_replace(source, pattern replacement)** function replaces only the first matched substring. 4. If the value of **flags** is **''** or null, the return value of **regexp_replace(source, pattern, replacement, flags)** is the character string after replacement.
compat_cursor	Determines the compatibility behavior of implicit cursor states. If this parameter is set and the O compatibility mode is used, the effective scope of implicit cursor states (**SQL%FOUND**, **SQL%NOTFOUND**, **SQL%ISOPNE**, and **SQL%ROWCOUNT**) are extended from only the currently executed function to all subfunctions invoked by this function.
proc_outparam_override	Determines the overloading of output parameters of a stored procedure. After this parameter is enabled, the stored procedure can be properly created and invoked even if only the output parameters of the stored procedure are different. Currently, this parameter can be used only when gsql and JDBC are used to connect to the database. If this parameter is enabled for other tools to connect to the database, stored procedures with the output parameters cannot be invoked.

Configuration Item	Behavior
proc_implicit_for_loop_variable	Determines the behavior of the **FOR_LOOP** query statement in a stored procedure. When this parameter is set, if **rec** has been defined in the **FOR rec IN query LOOP** statement, the defined **rec** variable is not reused and a new variable is created. Otherwise, the defined **rec** variable is reused and no new variable is created.
allow_procedure_compile_check	Determines the compilation check of the **SELECT** and **OPEN CURSOR** statements in a stored procedure. If this parameter is set, when the **SELECT**, **OPEN CURSOR FOR**, **CURSOR %rowtype**, or **for rec in** statement is executed in a stored procedure, the stored procedure cannot be created if the queried table does not exist, and the compilation check of the trigger function is not supported. If the queried table exists, the stored procedure is successfully created. Note: When creating a function in the encrypted state, you need to disable **allow_procedure_compile_check**.
char_coerce_compat	Determines the behavior when char(n) types are converted to other variable-length string types. By default, spaces at the end are omitted when the char(n) type is converted to other variable-length string types. After this parameter is enabled, spaces at the end are not omitted during conversion. In addition, if the length of the char(n) type exceeds the length of other variable-length string types, an error is reported. This parameter is valid only when the **sql_compatibility** parameter is set to **A**. After this parameter is enabled, spaces at the end are not omitted in implicit conversion, explicit conversion, or conversion by calling the **text(bpchar)** function.
truncate_numeric_tail_zero	Configuration item for numeric display. If this parameter is not set, numeric data is displayed in the default precision. When this parameter is set, the trailing zeros after the decimal point is hidden in all numeric output scenarios except to_char(numeric, format).
plsql_security_definer	After this parameter is enabled, the definer permission is used by default when a stored procedure is created.
array_count_compat	Controls the array.count function. If the parameter is enabled, the function returns **0**. Otherwise, the function returns **NULL**.
disable_emptystr2null	Disables the function of converting an empty string to null by default for the text, clob, blob, and raw character string types.

plsql_compile_check_options

Parameter description: Specifies database compatibility behavior. Multiple items are separated by commas (,).

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string

Default value: ""

NOTE

- Currently, only compatibility configuration items in **Table 17-5** are supported.
- Multiple items are separated by commas (,), for example, **set plsql_compile_check_options='for_loop,outparam';**.

Table 17-6 Compatibility configuration items

Configuration Item	Behavior
for_loop	Determines the behavior of the **FOR_LOOP** query statement in a stored procedure. When this parameter is set, if **rec** has been defined in the **FOR rec IN query LOOP** statement, the defined **rec** variable is not reused and a new variable is created. Otherwise, the defined **rec** variable is reused and no new variable is created. (It is the same as **proc_implicit_for_loop_variable** and will be incorporated later.)
outparam	When the output parameter overloading condition is met, the output parameters are checked. If the output parameters are constant, an error is reported.

a_format_version

Parameter description: Specifies the database platform compatibility configuration item. The value of this parameter is an enumerated string.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string

Default value: ""

NOTE

- Currently, only compatibility configuration items in **Table 17-5** are supported.
- Set a character string for the compatibility configuration item, for example, **set a_format_version='10c'**.

Table 17-7 Compatibility configuration items

Configuration Item	Compatibility Behavior Control
10c	Compatible version of platform A

a_format_dev_version

Parameter description: Specifies the database platform minor version compatibility configuration item. The value of this parameter is an enumerated string.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string

Default value: ""

NOTE

- Currently, only compatibility configuration items in **Table 17-5** are supported.
- Set a character string for the compatibility configuration item, for example, **set a_format_dev_version='s1'**.

Table 17-8 Compatibility configuration items

Configuration Item	Compatibility Behavior Control
s1	Compatible minor version of platform A, which affects functions: (TRUNC(date, fmt),ROUND(date, fmt),NVL2,LPAD,RPAD,ADD_MONTHS,MONTHS_BETWEEN,REGEXP_REPLACE ,REGEXP_COUNT,TREAT,EMPTY_CLOB,INSTRB) After this parameter is enabled, the int casted from text can be rounded off.

plpgsql.variable_conflict

Parameter description: Sets the priority of using stored procedure variables and table columns with the same name.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-2**.

Value range: a string

- **error** indicates that a compilation error is reported when the name of a stored procedure variable is the same as that of a table column.
- **use_variable** indicates that if the name of a stored procedure variable is the same as that of a table column, the variable is used preferentially.
- **use_column** indicates that if the name of a stored procedure variable is the same as that of a table column, the column name is used preferentially.

Default value: **error**

td_compatible_truncation

Parameter description: Specifies whether to enable features compatible with a Teradata database. You can set this parameter to **on** when connecting to a database compatible with the Teradata database, so that when you perform the **INSERT** operation, overlong strings are truncated based on the allowed maximum length before being inserted into char- and varchar-type columns in the target table. This ensures all data is inserted into the target table without errors reported.

NOTE

The string truncation function cannot be used if the **INSERT** statement includes a foreign table.

If inserting multi-byte character data (such as Chinese characters) to database with the character set byte encoding (such as SQL_ASCII or LATIN1), and the character data crosses the truncation position, the string is truncated based on its bytes instead of characters. Unexpected result will occur in tail after the truncation. If you want correct truncation result, you are advised to adopt encoding set such as UTF8, which has no character data crossing the truncation position.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-2**.

Value range: Boolean

- **on** indicates that overlong strings are truncated.
- **off** indicates that overlong strings are not truncated.

Default value: **off**

uppercase_attribute_name

Parameter description: Specifies whether to return column names in uppercase to the client. This parameter is used only in the ORA-compatible mode and centralized environment.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-2**.

Value range: Boolean

- **on** indicates that column names are returned to the client in uppercase.
- **off** indicates that column names are not returned to the client in uppercase.

Default value: **off**

lastval_supported

Parameter description: Specifies whether the lastval function can be used.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the lastval function can be used and the nextval function cannot be pushed down.
- **off** indicates that the lastval function cannot be used and the nextval function can be pushed down.

Default value: **off**

17.16 Fault Tolerance

This section describes parameters used for controlling how the server processes an error occurring in the database system.

exit_on_error

Parameter description: If this function is enabled, errors of the ERROR level will be upgraded to PANIC errors, and core stacks will be generated. It is mainly used to locate problems and test services.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-2**.

Value range: Boolean

- **on** indicates that errors of the ERROR level will be upgraded to PANIC errors.
- **off** indicates that errors of the ERROR level will not be upgraded.

Default value: **off**

restart_after_crash

Parameter description: If this parameter is set to **on** and a backend process crashes, GaussDB automatically reinitializes the backend process.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-2**.

Value range: Boolean

- **on** maximizes the availability of the database.

  In some circumstances (for example, when a management tool, such as xCAT, is used to manage GaussDB), setting this parameter to **on** maximizes the availability of the database.
- **off** indicates that a management tool is enabled to obtain control permission and take proper measures when a backend process crashes.

Default value: **on**

omit_encoding_error

Parameter description: If this parameter is set to **on** and the client character set of the database is encoded in UTF-8 format, character encoding conversion errors will be recorded in logs. Additionally, converted characters that have conversion errors will be ignored and replaced with question marks (?).

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-2**.

Value range: Boolean

- **on** indicates that characters that have conversion errors will be ignored and replaced with question marks (?), and error information will be recorded in logs.
- **off** indicates that characters that have conversion errors cannot be converted and error information will be directly displayed.

Default value: **off**

max_query_retry_times

The current feature is a lab feature. Contact Huawei technical support before using it.

Parameter description: Specifies the maximum number of times that an **SQL** statement with errors can be re-executed. **SQL** statements with the following types of errors can be re-executed: **Connection reset by peer**, **Lock wait timeout**, and **Connection timed out**. If this parameter is set to **0**, the re-execution function is disabled.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-2**.

Value range: an integer ranging from 0 to 20

Default value: **0**

cn_send_buffer_size

Parameter description: Specifies the size of the data buffer used for data transmission on the primary database node.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-2**.

Value range: an integer ranging from 8 to 128. The unit is KB.

Default value: **8KB**

retry_ecode_list

Parameter description: Specifies the list of SQL error types that support automatic retries.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-2**.

Value range: a string

Default value: **YY001 YY002 YY003 YY004 YY005 YY006 YY007 YY008 YY009 YY010 YY011 YY012 YY013 YY014 YY015 53200 08006 08000 57P01 XX003 XX009 YY016**

data_sync_retry

Parameter description: Specifies whether to keep running the database when updated data fails to be written into disks by using the **fsync** function. In some OSs, no error is reported even if **fsync** fails after the second attempt. As a result, data is lost.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-2**.

Value range: Boolean

- **on** indicates that the database keeps running and **fsync** is executed again after **fsync** fails.
- **off** indicates that a PANIC-level error is reported and the database is stopped after **fsync** fails.

Default value: **off**

remote_read_mode

Parameter description: Specifies whether to enable the remote read function. This function allows pages on the standby node to be read when reading pages on the primary node fails.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: enumerated values

- **off** indicates that the remote read function is disabled.
- **non_authentication** indicates that the remote read function is enabled but certificate authentication is not required.
- **authentication** indicates that the remote read function is enabled and certificate authentication is required.

Default value: **authentication**

17.17 Connection Pool Parameters

When a connection pool is used to access the database, database connections are established and then stored in the memory as objects during system running. When you need to access the database, no new connection is established. Instead, an existing idle connection is selected from the connection pool. After you finish accessing the database, the database does not disable the connection but puts it back into the connection pool. The connection can be used for the next access request.

cache_connection

Parameter description: Specifies whether to reclaim the connections of a connection pool.

This parameter is a **SIGHUP** parameter. Set it based on instructions provided in **Table 10-2**.

Value range: Boolean

- **on** indicates that the connections of a connection pool will be reclaimed.
- **off** indicates that the connections of a connection pool will not be reclaimed.

Default value: **on**

17.18 Transaction

This section describes the settings and value ranges of database transaction parameters.

transaction_isolation

Parameter description: specifies the isolation level of the current transaction.

This parameter is a **USERSET** parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string of case-sensitive characters. The values include:

- **serializable**: This value is equivalent to REPEATABLE READ in GaussDB.
- **read committed** indicates that only the data in committed transactions will be read.
- **repeatable read** indicates that only the data committed before transaction start is read. Uncommitted data or data committed in other concurrent transactions cannot be read.
- **default**: The value is the same as that of **default_transaction_isolation**.

Default value: read committed

transaction_read_only

Parameter description: Specifies whether the current transaction is a read-only transaction.

This parameter has a fixed value **on** during database restoration or on the standby node. Otherwise, set this parameter to the value of **default_transaction_read_only**.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the current transaction is a read-only transaction.
- **off** indicates that the current transaction can be a read/write transaction.

Default value: **off**

xc_maintenance_mode

Parameter description: Specifies whether the system is in maintenance mode.

This parameter is a SUSET parameter. Set it based on method 3 in **Table 10-1**.

Value range: Boolean

- **on** indicates that the system is in maintenance mode.
- **off** indicates that the system is not in maintenance mode.

NOTICE

Enable the maintenance mode with caution to avoid database data inconsistencies.

Default value: **off**

allow_concurrent_tuple_update

Parameter description: Specifies whether to allow concurrent update.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the concurrent update is allowed.
- **off** indicates that the system is not in maintenance mode.

Default value: **on**

transaction_deferrable

Parameter description: Specifies whether to delay the execution of a read-only serial transaction without incurring an execution failure. Assume this parameter is set to **on**. When the server detects that the tuples read by a read-only transaction are being modified by other transactions, it delays the execution of the read-only transaction until the other transactions finish modifying the tuples. This parameter is reserved and does not take effect in this version. Similar to this parameter, the **default_transaction_deferrable** parameter is used to specify whether to allow delayed execution of a transaction.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the execution of a read-only serial transaction can be delayed.
- **off** indicates that the execution of a read-only serial transaction cannot be delayed.

Default value: **off**

enable_show_any_tuples

Parameter description: This parameter is available only in a read-only transaction and is used for analysis. When this parameter is set to **on** or **true**, all versions of tuples in the table are displayed.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** or **true** indicates that all versions of tuples in the table are displayed.
- **off** or **false** indicates that no versions of tuples in the table are displayed.

Default value: **off**

replication_type

Parameter description: Specifies whether the current HA mode is standalone, primary/standby/secondary, or one primary multiple standbys.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

This parameter is an internal parameter. Do not set it.

Value range: 0 to 2

- **2** Indicates the single primary mode. In this mode, the standby node cannot be expanded.
- **1** Indicates that the one-primary-multiple-standby mode is used, covering all scenarios. This mode is recommended.
- **0** Indicates the primary/standby mode. Currently, this mode is not supported.

Default value: **1**

pgxc_node_name

Parameter description: Specifies the name of a node.

This parameter is a POSTMASTER parameter. Set it based on **Table 10-2**.

When a standby node requests to replicate logs on the primary node, if the **application_name** parameter is not set, the **pgxc_node_name** parameter is used as the name of the streaming replication slot of the standby node on the primary node. The streaming replication slot is named in the following format: Value of this parameter_IP address of the standby node_Port number of the standby node. The IP address and port number of the standby node are obtained from the IP address and port number of the standby node specified by the **replconninfo** parameter. The maximum length of a streaming replication slot name is 61 characters. If the length of the concatenated string exceeds 61 characters, the truncated **pgxc_node_name** will be used for concatenation to ensure that the length of the streaming replication slot name is less than or equal to 61 characters.

CAUTION

After this parameter is modified, the database instance will fail to be connected. You are advised not to modify this parameter.

Value range: a string

Default value: current node name

enable_defer_calculate_snapshot

Parameter description: Specifies the delay in calculating **xmin** and **oldestxmin**. Calculation is triggered only when 1000 transactions are executed or the interval is 1s. If this parameter is set to **on**, the overhead of calculating snapshots can be reduced in heavy-load scenarios, but the progress of **oldestxmin** is slow, affecting tuple recycling. If this parameter is set to **off**, **xmin** and **oldestxmin** can be calculated in real time, but the overhead for calculating snapshots increases.

This parameter is a SIGHUP parameter. Set it based on **Table 10-2**.

Value range: Boolean

- **on** indicates that snapshots **xmin** and **oldestxmin** are calculated with a delay.
- **off** indicates that snapshot **xmin** and **oldestxmin** are calculated in real time.

Default value: **on**

17.19 Replication Parameters of Two Database Instances

RepOriginId

Parameter description: This parameter is a session-level GUC parameter. In bidirectional logical replication, set it to a non-zero value to avoid infinite data replication.

This parameter is a USERSET parameter. Set it based on Method 3 provided in **Table 10-1**.

Value range: an integer ranging from 0 to 2147483647

Default value: **0**

stream_cluster_run_mode

Parameter description: Specifies whether a DN is in the primary or standby instance in dual-instance streaming DR scenarios. For single-instance scenarios, the DN is in the primary instance by default.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: enumerated values

- **cluster_primary** indicates that the node is in the primary instance.
- **cluster_standby** indicates that the node is in the standby instance.

Default value: **cluster_primary**

enable_roach_standby_cluster

Parameter description: Sets the standby database instances to read-only in dual-database instance mode. Only users with the sysadmin permission can access this parameter.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the read-only mode is enabled for the standby database instances.
- **off** indicates that the read-only mode is disabled for the standby database instances. In this case, the standby database instances can be read and written.

Default value: **off**

17.20 Developer Options

allow_system_table_mods

Parameter description: Specifies whether the structure of a system catalog or the name of a system schema can be modified.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the structure of the system catalog or the name of the system schema can be modified.
- **off** indicates that the structure of the system catalog or the name of the system schema cannot be modified.

Default value: **off**

CAUTION

You are not advised to change the default value of this parameter. If this parameter is set to **on**, system tables may be damaged and the database may fail to be started.

allow_create_sysobject

Parameter description: Specifies whether objects such as functions, stored procedures, synonyms, aggregate functions, and operators can be created or modified in the system schema. The system schema refers to the schema provided by the database after initialization, excluding the public schema. The OID of the system schema is usually smaller than 16384.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the initial user and system administrator can create or modify objects such as functions, stored procedures, synonyms, and aggregate functions in the system schema, and the initial user can create operators in the system schema. For details about whether other users are allowed to create these objects, see the permission requirements of the corresponding schema.
- **off** indicates that all users are not allowed to create or modify objects such as functions, stored procedures, synonyms, aggregate functions, and operators in the system schema.

Default value: **on**

debug_assertions

Parameter description: Specifies whether to enable various assertion checks. This parameter assists in debugging. If you are experiencing strange problems or crashes, set this parameter to **on** to identify programming defects. To use this parameter, the macro USE_ASSERT_CHECKING must be defined (through the configure option **--enable-cassert**) during the GaussDB compilation.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that various assertion checks are enabled.
- **off** indicates that various assertion checks are disabled.

NOTE

If you compile GaussDB with the assertion check enabled, this parameter is set to **on** by default.

Default value: **off**

ignore_checksum_failure

Parameter description: Specifies whether to ignore check failures (but still generates an alarm) and continues reading data. Continuing reading data may result in breakdown, damaged data being transferred or hidden, failure of data recovery from remote nodes, or other serious problems. You are not advised to modify the settings.

This parameter is a SUSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that data check errors are ignored.
- **off** indicates that data check errors are reported.

Default value: **off**

ignore_system_indexes

Parameter description: Specifies whether to ignore system indexes when reading system catalog (but still update the indexes when modifying the tables).

This parameter is a BACKEND parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

This parameter is useful for recovering data from tables whose system indexes are damaged.

Value range: Boolean

- **on** indicates that system indexes are ignored.
- **off** indicates that system indexes are not ignored.

Default value: **off**

post_auth_delay

Parameter description: Specifies the delay in the connection to the server after a successful authentication. Developers can attach a debugger to the server startup process.

This parameter is a BACKEND parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 2147. The unit is s.

Default value: **0**

NOTE

This parameter is used only for commissioning and fault locating. To prevent impact on service running, ensure that the default value **0** is used in the production environment. If this parameter is set to a value other than 0, the cluster status may be abnormal due to a long authentication delay.

pre_auth_delay

Parameter description: Specifies the period of delaying authentication after the connection to the server is started. Developers can attach a debugger to the authentication procedure.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 60. The unit is s.

Default value: **0**

NOTE

This parameter is used only for commissioning and fault locating. To prevent impact on service running, ensure that the default value **0** is used in the production environment. If this parameter is set to a value other than 0, the cluster status may be abnormal due to a long authentication delay.

trace_notify

Parameter description: Specifies whether to enable the function of generating debugging output for the **LISTEN** and **NOTIFY** commands. The level of **client_min_messages** or **log_min_messages** must be **debug1** or lower so that debugging output can be recorded in the client or server logs, respectively.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the function is enabled.
- **off** indicates that the function is disabled.

Default value: **off**

trace_recovery_messages

Parameter description: Specifies whether to enable logging of recovery-related debugging output. This parameter allows users to overwrite the normal setting of **log_min_messages**, but only for specific messages. This is intended for the use in debugging the standby node.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: enumerated values. Valid values include **debug5**, **debug4**, **debug3**, **debug2**, **debug1**, and **log**. For details about the parameter values, see **log_min_messages**.

Default value: **log**

NOTE

- **log** indicates that recovery-related debugging information will not be logged.
- Except the default value **log**, each of the other values indicates that recovery-related debugging information at the specified level will also be logged. Common settings of **log_min_messages** enable logs to be unconditionally recorded into server logs.

trace_sort

Parameter description: Specifies whether to print information about resource usage during sorting operations. This parameter is available only when the macro TRACE_SORT is defined during the GaussDB compilation. However, TRACE_SORT is currently defined by default.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the function is enabled.
- **off** indicates that the function is disabled.

Default value: **off**

zero_damaged_pages

Parameter description: Specifies whether to detect a damaged page header that causes GaussDB to report an error, aborting the current transaction.

This parameter is a SUSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

Setting this parameter to **on** causes the system to report a warning, zero out the damaged page, and continue processing. This behavior will destroy data, including all the rows on the damaged page. However, it allows you to bypass the error and retrieve rows from any undamaged pages that may be present in the table. Therefore, it is useful for restoring data if corruption has occurred due to a hardware or software error. In most cases, you are advised not to set this parameter to **on** if you want to restore data from damaged pages.

Default value: **off**

remotetype

Parameter description: Specifies the remote connection type.

This parameter is a BACKEND parameter. Set it based on instructions provided in **Table 10-1**.

Value range: enumerated values. Valid values are **application**, **datanode**, and **internaltool**.

Default value: **application**

max_user_defined_exception

Parameter description: Specifies the maximum number of exceptions. The default value cannot be changed.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer. Currently, only the fixed value **1000** is supported.

Default value: **1000**

enable_fast_numeric

Parameter description: Specifies whether to enable optimization for numeric data calculation. Calculation of numeric data is time-consuming. Numeric data is converted into int64- or int128-type data to improve numeric data calculation performance.

This parameter is a SUSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** or **true** indicates that optimization for numeric data calculation is enabled.
- **off** or **false** indicates that optimization for numeric data calculation is disabled.

Default value: **on**

enable_compress_spill

Parameter description: Specifies whether to enable the compression function of writing data to disk.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** or **true** indicates that optimization for writing data to disk is enabled.
- **off** or **false** indicates that optimization for writing data to a disk is disabled.

Default value: **on**

resource_track_log

Parameter description: Specifies the log level of self-diagnosis. Currently, this parameter takes effect only in multi-column statistics. (The current feature is a lab feature. Contact Huawei technical support before using it.)

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string

- **summary**: Brief diagnosis information is displayed.
- **detail**: Detailed diagnosis information is displayed.

Currently, the two parameter values differ only when there is an alarm about multi-column statistics not collected. If the parameter is set to **summary**, such an alarm will not be displayed. If it is set to **detail**, such an alarm will be displayed.

Default value: **summary**

show_acce_estimate_detail

Parameter description: The evaluation information is generally used by O&M personnel during maintenance, and it may affect the output display of the **EXPLAIN** statement. Therefore, this parameter is disabled by default. The evaluation information is displayed only if the **verbose** option of the **EXPLAIN** statement is enabled. (Due to specification changes, the current version no longer supports the current feature. Do not use this feature.)

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the evaluation information is displayed in the output of the **EXPLAIN** statement.
- **off** indicates that the evaluation information is not displayed in the output of the **EXPLAIN** statement.

Default value: **off**

NOTE

The current version does not support database acceleration. Therefore, this parameter does not take effect after being set.

support_batch_bind

Parameter description: Specifies whether to batch bind and execute PBE statements through interfaces such as JDBC, ODBC, and libpq.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that batch binding and execution are used.
- **off** indicates that batch binding and execution are not used.

Default value: **on**

numa_distribute_mode

Parameter description: Specifies the distribution of some shared data and threads among NUMA nodes. This parameter is used to optimize the performance of large-scale ARM servers with multiple NUMA nodes. Generally, you do not need to set this parameter.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string. The valid values are **none** and **all**.

- **none** indicates that this function is disabled.
- **all** indicates that some shared data and threads are distributed to different NUMA nodes to reduce the number of remote access times and improve performance. Currently, this function applies only to ARM servers with multiple NUMA nodes. All NUMA nodes must be available for database processes. You cannot select only some NUMA nodes.

NOTE

In the current version, **numa_distribute_mode** cannot be set to **all** on the x86 architecture.

Default value: **none**

log_pagewriter

Parameter description: Specifies whether to display the page refresh information of a thread and details about an incremental check point after the incremental check point is enabled. You are not advised to set this parameter to **true** because a large amount of information will be generated.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

Default value: **on**

advance_xlog_file_num

Parameter description: Specifies the number of Xlog files that are periodically initialized in advance in the background. This parameter is used to prevent the Xlog file initialization from affecting the performance during transaction submission. However, such a fault may occur only when the system is overloaded. Therefore, you do not need to set this parameter.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 1000000. The value **0** indicates that initialization is not performed in advance. For example, the value **10** indicates that the background thread periodically initializes 10 Xlog files in advance based on the write location of the current Xlog.

Default value: **0**

enable_beta_opfusion

Parameter description: Specifies whether to accelerate the execution of SQL statements, such as aggregate functions and sorting in TPC-C when **enable_opfusion** is set to **on**.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on**: enabled.
- **off**: disabled.

Default value: **off**

enable_csqual_pushdown

Parameter description: Specifies whether to deliver filter criteria for a rough check during query.

This parameter is a SUSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that a rough check is performed with filter criteria delivered during query.
- **off** indicates that a rough check is performed without filter criteria delivered during query.

Default value: **on**

string_hash_compatible

Parameter description: Specifies whether to use the same method to calculate char-type hash values and varchar- or text-type hash values. Based on the setting of this parameter, you can determine whether a redistribution is required when a distribution column is converted from a char-type data distribution into a varchar- or text-type data distribution.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the same calculation method is used and a redistribution is not required.
- **off** indicates that different calculation methods are used and a redistribution is required.

NOTE

Calculation methods differ in the length of input strings used for calculating hash values. (For a char-type hash value, spaces following a string are not counted as the length. For a text- or varchar-type hash value, the spaces are counted.) The hash value affects the calculation result of queries. To avoid query errors, do not modify this parameter during database running once it is set.

Default value: **off**

pldebugger_timeout

Parameter description: Specifies the timeout interval for the pldebugger server to wait for a response from the debug end.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 1 to 86400. The unit is second.

Default value: **15min**

plsql_show_all_error

Parameter description: Specifies whether to skip errors and continue compiling PLPGSQL objects.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

Default value: **off**

ustore_attr

Parameter description: This parameter is used to control the information statistics of Ustore tables, rollback type, and data verification during the running of key modules (including data, indexes, rollback segments, and playback). This parameter helps R&D engineers locate faults.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string. This parameter is set in key-value mode. The mapping between keys and values is as follows: If multiple key-value pairs are used, use semicolons (;) to separate them. For example:

ustore_attr='ustore_verify_level=NORMAL;ustore_verify_module=UPAGE:UBTREE:UNDO:REDO'.

NOTE

When setting **ustore_attr**, do not leave spaces or other characters before and after the equal sign (=) between key and value, for example, **ustore_attr='ustore_verify_level = NORMAL**. Otherwise, the parameter is invalid during kernel code verification and the parameter setting fails.

- **ustore_verify_level**: verification level.

  Value range: a string. For details, see the following table.

  Table 17-9 Parameter value meaning of ustore_verify_level

Parameter Value	Description
FAST	**FAST** indicates fast verification. A few content is verified and the impact on performance is minimized.
NORMAL	**NORMAL** indicates that the verification is routinely performed. Compared with the fast verification, this verification involves more content, and the impact on the performance is medium.
SLOW	**SLOW** indicates slow verification. The verification content is the largest, and the performance is greatly affected.

  Default value: an empty string

- **ustore_verify_module**: module that controls verification.

  Value range: a string of case-insensitive characters. The value can be one or more of **UPAGE**, **UBTREE**, **UNDO**, and **REDO**, or it can be **ALL** or **NULL**. When multiple values of **UPAGE**, **UBTREE**, **UNDO**, and **REDO** are used, separate them with colons (:).

  For example, **ustore_verify_module=UPAGE:UBTREE:UNDO:REDO**.

  Table 17-10 Parameter value meaning of ustore_verify_module

Parameter Value	Description
UPAGE	Indicates that data page verification is enabled.

Parameter Value	Description
UBTREE	Indicates that UBtree index verification is enabled.
UNDO	Indicates that rollback segment data verification is enabled.
REDO	Indicates that data page verification for the REDO process is enabled.
ALL	Indicates that data verification is enabled for the UPAGE, UBTREE, UNDO, and REDO modules.
NULL	Indicates that data verification for the UPAGE, UBTREE, UNDO, and REDO modules is disabled.

Default value: an empty string

- **index_trace_level**: specifies whether to enable index tracing and control the printing level. After this function is enabled, information about index tuples that meet the conditions is printed based on the printing level during index scanning.

  Value range: a string. The values are described in the following table.

  Default value: **NO**

Table 17-11 Parameter value meaning of index_trace_level

Parameter Value	Description
NO	No additional information is printed.
NORMAL	Information about visible index tuples is printed, including: ● ID and offset of the index page where the current index tuple is located ● Current tuple status ● TID and partOid corresponding to the current tuple ● xmin and xmax information corresponding to the current tuple ● Current tuple content (if **enable_log_tuple** is set to **on**).
VISIBILITY	On the basis of **NORMAL**, the information about the index tuples that do not pass the visibility check is printed and whether the index tuples are visible is marked.
SHOWHIKEY	On the basis of **VISIBILITY**, the system tries to print the information about the HIKEY tuple on the page.

Parameter Value	Description
ALL	Information about all tuples on the scanned index page is printed.

0

- **enable_log_tuple**: specifies whether to print the contents of related tuples when printing log-level prompts for troubleshooting and locating.

  Value range: **on** or **off** (case-insensitive)

  Default value: **off**

  Note: This parameter has been discarded.
- **enable_ustore_sync_rollback**: indicates whether to enable synchronous rollback for Ustore tables.

  Value range: Boolean

  Default value: **true**
- **enable_ustore_async_rollback**: indicates whether to enable asynchronous rollback for Ustore tables.

  Value range: Boolean

  Default value: **true**
- **enable_ustore_page_rollback**: indicates whether to enable page rollback for Ustore tables.

  Value range: Boolean

  Default value: **true**
- **enable_ustore_partial_seqscan**: indicates whether to enable partial scan for Ustore tables.

  Value range: Boolean

  Default value: **false**
- **enable_candidate_buf_usage_count**: indicates whether to enable buffer usage statistics.

  Value range: Boolean

  Default value: **false**
- **ustats_tracker_naptime**: specifies the interval for collecting statistics on Ustore tables.

  Value range: [1,INT_MAX/1000]

  Default value: **20**, in seconds
- **umax_search_length_for_prune**: specifies the maximum search depth of the prune operation on the Ustore table.

  Value range: [1,INT_MAX/1000]

  Default value: **10** (times)

Default value: an empty string

NOTE

This parameter applies only to Ustore tables in a centralized system or a standalone openGauss system.

17.21 Auditing

17.21.1 Audit Switch

audit_enabled

Parameter description: Specifies whether to enable or disable the audit process. After the audit process is enabled, the auditing information written by the background process can be read from the pipe and written into audit files.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the auditing function is enabled.
- **off** indicates that the auditing function is disabled.

Default value: **on**

audit_directory

Parameter description: Specifies the storage directory of audit files. A path relative to the **data** directory. Only the sysadmin user can access this parameter.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string

Default value: pg_audit If **om** is used for database deployment, audit logs are stored in *$GAUSSLOG*/**pg_audit/***Instance name*.

NOTICE

- You need to set different audit file directories for different DNs. Otherwise, audit logs will be abnormal.
- If the value of **audit_directory** in the configuration file is an invalid path, the audit function cannot be used.

NOTE

- Valid path: Users must have read and write permissions on the path.
- Invalid path: Users do not have read or write permissions on an invalid path.

audit_data_format

Parameter description: Audits the format of log files. Currently, only the binary format is supported. Only the sysadmin user can access this parameter.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string

Default value: **binary**

audit_rotation_interval

Parameter description: Specifies the interval of creating an audit log file. If the difference between the current time and the time when the previous audit log file is created is greater than the value of **audit_rotation_interval**, a new audit log file will be generated.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 1 to *INT_MAX*/60. The unit is min.

Default value: 1d

NOTICE

Adjust this parameter only when required. Otherwise, **audit_resource_policy** may fail to take effect. To control the storage space and time of audit logs, set the **audit_resource_policy**, **audit_space_limit**, and **audit_file_remain_time** parameters.

audit_rotation_size

Parameter description: Specifies the maximum capacity of an audit log file. If the total number of messages in an audit log exceeds the value of **audit_rotation_size**, the server will generate a new audit log file.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 1024 to 1048576. The unit is KB.

Default value: **10MB**

NOTICE

- Do not adjust this parameter unless necessary. Otherwise, **audit_resource_policy** may fail to take effect. To control the storage space and time of audit logs, set the **audit_resource_policy**, **audit_space_limit**, and **audit_file_remain_time** parameters.
- If the space occupied by a single record in an audit log file exceeds the value of this parameter, the log file is regarded as an invalid log file.

audit_resource_policy

Parameter description: Specifies the policy for determining whether audit logs are preferentially stored by space or time.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that audit logs are preferentially stored by space. A maximum of **audit_space_limit** logs can be stored.
- **off** indicates that audit logs are preferentially stored by time. A minimum duration of **audit_file_remain_time** logs must be stored.

Default value: **on**

audit_file_remain_time

Parameter description: Specifies the minimum duration required for recording audit logs. This parameter is valid only when **audit_resource_policy** is set to **off**.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 730. The unit is day. **0** indicates that the storage duration is not limited.

Default value: **90**

audit_space_limit

Parameter description: Specifies the total disk space occupied by audit files.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 1024 KB to 1024 GB. The unit is KB.

Default value: **1GB**

NOTICE

In the multi-audit thread scenario, the minimum disk space occupied by audit files is the product of values of **audit_thread_num** and **audit_rotation_size**. If the value of this parameter is too small, the disk space occupied by audit files may exceed the value of this parameter.

audit_file_remain_threshold

Parameter description: Specifies the maximum number of audit files in the audit directory.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 100 to 1048576

Default value: **1048576**

NOTICE

- Ensure that this parameter is set to **1048576**. Do not adjust this parameter unless necessary. Otherwise, **audit_resource_policy** may fail to take effect. To control the storage space and time of audit logs, set the **audit_resource_policy**, **audit_space_limit**, and **audit_file_remain_time** parameters.
- In the multi-audit thread scenario, do not adjust this parameter unless necessary. Ensure that the value of this parameter is greater than or equal to the value of **audit_thread_num**. Otherwise, the audit function cannot be used and the database is abnormal.

audit_thread_num

Parameter description: Specifies the number of audit threads.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 1 to 48

Default value: **1**

NOTICE

- When **audit_dml_state** is enabled and high performance is required, you are advised to increase the value of this parameter to ensure that audit messages can be processed and recorded in a timely manner.
- Ensure that the value of this parameter is less than or equal to the maximum number of audit files in the audit directory (**audit_file_remain_threshold**). Otherwise, the audit function cannot be used and the database is abnormal.

17.21.2 User and Permission Audit

audit_login_logout

Parameter description: Specifies whether to audit the GaussDB user's login (including login success and failure) and logout.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 7

- **0** indicates that the function of auditing users' logins and logouts is disabled.
- **1** indicates that only successful user logins are audited.
- **2** indicates that only failed user logins are audited.

- **3** indicates that successful and failed user logins are audited.
- **4** indicates that only user logouts are audited.
- **5** indicates that successful user logouts and logins are audited.
- **6** indicates that failed user logouts and logins are audited.
- **7** indicates that successful user logins, failed user logins, and logouts are audited.

Default value: **7**

audit_database_process

Parameter description: Specifies whether to audit GaussDB start, stop, recovery, and switchover operations.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: 0 or **1**

- **0** indicates that the function of auditing GaussDB start, stop, recovery, and switchover operations is disabled.
- **1** indicates that the function of auditing GaussDB start, stop, recovery, and switchover operations is enabled.

Default value: **1**

NOTICE

When the database is started, the standby DN is promoted to primary. Therefore, the DN type in the audit log is **system_switch** when the DN is started.

audit_user_locked

Parameter description: Specifies whether to audit the GaussDB user's locking and unlocking.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: 0 or **1**

- **0** indicates that the function of auditing user's locking and unlocking is disabled.
- **1** indicates that the function of auditing user's locking and unlocking is enabled.

Default value: **1**

audit_user_violation

Parameter description: Specifies whether to audit the access violation operations of a user.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: 0 or **1**

- **0** indicates that the function of auditing the access violation operations of a user is disabled.
- **1** indicates that the function of auditing the access violation operations of a user is enabled.

Default value: **0**

audit_grant_revoke

Parameter description: Specifies whether to audit the granting and reclaiming of the GaussDB user's permission.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: 0 or **1**

- **0** indicates that the function of auditing the granting and reclaiming of a user's permission is disabled.
- **1** indicates that the function of auditing the granting and reclaiming of a user's permission is enabled.

Default value: **1**

17.21.3 Operation Audit

audit_system_object

Parameter description: Specifies whether to audit the CREATE, DROP, and ALTER operations on the GaussDB database object. The GaussDB database objects include databases, users, schemas, and tables. The operations on the database object can be audited by changing the value of this parameter.

This parameter is a **SIGHUP** parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 134217727

- **0** indicates that the function of auditing the CREATE, DROP, and ALTER operations on the GaussDB database object can be disabled.
- Other values indicate that the CREATE, DROP, and ALTER operations on a certain or some GaussDB database objects are audited.

Value description:

The value of this parameter is calculated by 27 binary bits. The 27 binary bits represent 27 types of GaussDB objects. If the corresponding binary bit is set to **0**, the CREATE, DROP, and ALTER operations on corresponding database objects are not audited. If it is set to **1**, the CREATE, DROP, and ALTER operations are audited. For details about the audit contents represented by these 27 binary bits, see **Table 17-12**.

When SQL patches are audited and **audit_dml_state_select** are enabled, an SQL patch operation will be audited twice and recorded as DML and DDL operations in the audit log, respectively.

Default value: **67121159**

Table 17-12 Meaning of each value for the **audit_system_object** parameter

Binary Bit	Description	Value Description
Bit 0	Whether to audit the CREATE, DROP, and ALTER operations on databases.	● **0** indicates that the CREATE, DROP, and ALTER operations on these objects are not audited. ● **1** indicates that the CREATE, DROP, and ALTER operations on these objects are audited.
Bit 1	Whether to audit the CREATE, DROP, and ALTER operations on schemas.	● **0** indicates that the CREATE, DROP, and ALTER operations on these objects are not audited. ● **1** indicates that the CREATE, DROP, and ALTER operations on these objects are audited.
Bit 2	Whether to audit the CREATE, DROP, and ALTER operations on users.	● **0** indicates that the CREATE, DROP, and ALTER operations on these objects are not audited. ● **1** indicates that the CREATE, DROP, and ALTER operations on these objects are audited.
Bit 3	Whether to audit the CREATE, DROP, ALTER, and TRUNCATE operations on tables.	● **0** indicates that the CREATE, DROP, ALTER, and TRUNCATE operations on these objects are not audited. ● **1** indicates that the CREATE, DROP, ALTER, and TRUNCATE operations on these objects are audited.
Bit 4	Whether to audit the CREATE, DROP, and ALTER operations on indexes.	● **0** indicates that the CREATE, DROP, and ALTER operations on these objects are not audited. ● **1** indicates that the CREATE, DROP, and ALTER operations on these objects are audited.
Bit 5	Whether to audit the CREATE and DROP operations on VIEW and MATVIEW objects.	● **0** indicates that the CREATE and DROP operations on these objects are not audited. ● **1** indicates that the CREATE and DROP operations on these objects are audited.

Binary Bit	Description	Value Description
Bit 6	Whether to audit the CREATE, DROP, and ALTER operations on triggers.	● **0** indicates that the CREATE, DROP, and ALTER operations on these objects are not audited. ● **1** indicates that the CREATE, DROP, and ALTER operations on these objects are audited.
Bit 7	Whether to audit the CREATE, DROP, and ALTER operations on procedures/ functions.	● **0** indicates that the CREATE, DROP, and ALTER operations on these objects are not audited. ● **1** indicates that the CREATE, DROP, and ALTER operations on these objects are audited.
Bit 8	Whether to audit the CREATE, DROP, and ALTER operations on tablespaces.	● **0** indicates that the CREATE, DROP, and ALTER operations on these objects are not audited. ● **1** indicates that the CREATE, DROP, and ALTER operations on these objects are audited.
Bit 9	Whether to audit the CREATE, DROP, and ALTER operations on resource pools.	● **0** indicates that the CREATE, DROP, and ALTER operations on these objects are not audited. ● **1** indicates that the CREATE, DROP, and ALTER operations on these objects are audited.
Bit 10	Whether to audit the CREATE, DROP, and ALTER operations on workloads.	● **0** indicates that the CREATE, DROP, and ALTER operations on these objects are not audited. ● **1** indicates that the CREATE, DROP, and ALTER operations on these objects are audited.
Bit 11	Reserved	-
Bit 12	Whether to audit the CREATE, DROP, and ALTER operations on data sources.	● **0** indicates that the CREATE, DROP, and ALTER operations on these objects are not audited. ● **1** indicates that the CREATE, DROP, and ALTER operations on these objects are audited.
Bit 13	Reserved	-

Binary Bit	Description	Value Description
Bit 14	Whether to audit the CREATE, DROP, and ALTER operations on ROW LEVEL SECURITY objects.	• **0** indicates that the CREATE, DROP, and ALTER operations on these objects are not audited. • **1** indicates that the CREATE, DROP, and ALTER operations on these objects are audited.
Bit 15	Whether to audit the CREATE, DROP, and ALTER operations on types.	• **0** indicates that the CREATE, DROP, and ALTER operations on types are not audited. • **1** indicates that the CREATE, DROP, and ALTER operations on types are audited.
Bit 16	Whether to audit the CREATE, DROP, and ALTER operations on text search objects (CONFIGURATION and DICTIONARY).	• **0** indicates that the CREATE, DROP, and ALTER operations on text search objects are not audited. • **1** indicates that the CREATE, DROP, and ALTER operations on text search objects are audited.
Bit 17	Whether to audit the CREATE, DROP, and ALTER operations on directories.	• **0** indicates that the CREATE, DROP, and ALTER operations on directories are not audited. • **1** indicates that the CREATE, DROP, and ALTER operations on directories are audited.
Bit 18	Whether to audit the CREATE, DROP, and ALTER operations on synonyms.	• **0** indicates that the CREATE, DROP, and ALTER operations on types are not audited. • **1** indicates that the CREATE, DROP, and ALTER operations on types are audited.
Bit 19	Whether to audit the CREATE, DROP, and ALTER operations on sequences.	• **0** indicates that the operations are not audited. • **1** indicates that the operations are audited.
Bit 20	Whether to audit the CREATE and DROP operations on CMK and CEK objects.	• **0** indicates that the CREATE and DROP operations on CMK and CEK objects are not audited. • **1** indicates that the CREATE and DROP operations on CMK and CEK objects are audited.

Binary Bit	Description	Value Description
Bit 21	Whether to audit the CREATE, DROP, and ALTER operations on PACKAGE objects.	● **0** indicates that the operations are not audited. ● **1** indicates that the operations are audited.
Bit 22	Whether to audit the CREATE and DROP operations on MODEL objects.	● **0** indicates that the CREATE and ALTER operations are not audited. ● **1** indicates that the CREATE and DROP operations are audited.
Bit 23	Whether to audit the CREATE, DROP, and ALTER operations on PUBLICATION and SUBSCRIPTION objects.	● **0** indicates that the CREATE, DROP, and ALTER operations are not audited. ● **1** indicates that the CREATE, DROP, and ALTER operations are audited.
Bit 24	Whether to audit the ALTER and DROP operations on the **gs_global_config** objects.	● **0** indicates that the ALTER and DROP operations are not audited. ● **1** indicates that the ALTER and DROP operations are audited.
Bit 25	Whether to audit the CREATE, DROP, and ALTER operations on FOREIGN DATA WRAPPER objects.	● **0** indicates that the CREATE, DROP, and ALTER operations are not audited. ● **1** indicates that the CREATE, DROP, and ALTER operations are audited.
Bit 26	Whether to audit the CREATE, ENABLE, DISABLE, and DROP operations on SQL patches.	● **0** indicates that the CREATE, ENABLE, DISABLE, and DROP operations on SQL patches are not audited. ● **1** indicates that the CREATE, ENABLE, DISABLE, and DROP operations on SQL patches are audited.

audit_dml_state

Parameter description: Specifies whether to audit the INSERT, UPDATE, and DELETE operations on a specific table.

This parameter is a **SIGHUP** parameter. Set it based on instructions provided in **Table 10-1**.

Value range: 0 or **1**

- **0** indicates that the function of auditing the DML operations (except SELECT) is disabled.

- **1** indicates that the function of auditing the DML operations (except SELECT) is enabled.

Default value: **0**

audit_dml_state_select

Parameter description: Specifies whether to audit the SELECT operation.

This parameter is a **SIGHUP** parameter. Set it based on instructions provided in **Table 10-1**.

Value range: 0 or **1**

- **0** indicates that the SELECT auditing function is disabled.
- **1** indicates that the SELECT auditing function is enabled.

Default value: **0**

audit_function_exec

Parameter description: Specifies whether to record the audit information during the execution of the stored procedures, anonymous blocks, or user-defined functions (excluding system functions).

This parameter is a **SIGHUP** parameter. Set it based on instructions provided in **Table 10-1**.

Value range: 0 or **1**

- **0** indicates that the function of auditing the procedure or function execution is disabled.
- **1** indicates that the function of auditing the procedure or function execution is enabled.

Default value: **0**

audit_copy_exec

Parameter description: Specifies whether to audit the COPY operation.

This parameter is a **SIGHUP** parameter. Set it based on instructions provided in **Table 10-1**.

Value range: 0 or **1**

- **0** indicates that the COPY auditing function is disabled.
- **1** indicates that the COPY auditing function is enabled.

Default value: **1**

audit_set_parameter

Parameter description: Specifies whether to audit the SET operation.

This parameter is a **SIGHUP** parameter. Set it based on instructions provided in **Table 10-1**.

Value range: 0 or **1**

- **0** indicates that the SET auditing function is disabled.
- **1** indicates that the SET auditing function is enabled.

Default value: **0**

audit_xid_info

Parameter description: Specifies whether to record the transaction ID of the SQL statement in the **detail_info** column of the audit log.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: 0 or **1**

- **0** indicates that the function of recording transaction IDs in audit logs is disabled.
- **1** indicates that the function of recording transaction IDs in audit logs is enabled.

Default value: **0**

NOTICE

If this function is enabled, the **detail_info** information in audit logs starts with *xid*. For example:

```
detail_info: xid=14619 , create table t1(id int);
```

If transaction IDs do not exist, *xid* is recorded as **NA** in audit logs.

enableSeparationOfDuty

Parameter description: Specifies whether the separation of three duties is enabled.

This parameter is a **POSTMASTER** parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the separation of three duties is enabled.
- **off** indicates that the separation of three duties is disabled.

Default value: **off**

enable_nonsysadmin_execute_direct

Parameter description: Specifies whether non-system administrators and non-monitor administrator are allowed to execute the EXECUTE DIRECT ON statement.

This parameter is a **POSTMASTER** parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that any user is allowed to execute the EXECUTE DIRECT ON statement.
- **off** indicates that only the system administrators and monitor administrators are allowed to execute the EXECUTE DIRECT ON statement.

Default value: **off**

enable_access_server_directory

Parameter description: Specifies whether to allow non-initial users to create, modify, and delete directories.

This parameter is a **SIGHUP** parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that non-initial users have the permission to create, modify, and delete directories.
- **off** indicates that non-initial users do not have the permission to create, modify, and delete directories.

Default value: **off**

NOTICE

- For security purposes, only the initial user can create, modify, and delete DIRECTORY objects by default.
- If **enable_access_server_directory** is enabled, users with the SYSADMIN permission and users who inherit the gs_role_directory_create permission of the built-in role can create directory objects. A user with the SYSADMIN permission, the owner of a directory, a user who is granted with the DROP permission for the directory, or a user who inherits the gs_role_directory_drop permission of the built-in role can delete a directory. A user with the SYSADMIN permission and the owner of a directory object can change the owner of the directory object, and the user must be a member of the new owner.

17.22 CM Parameters

Modifying CM parameters affects the running mechanism of GaussDB. You are advised to ask GaussDB engineers to do it for you. For details about how to modify the CM parameters, see method 1 in **Table 10-2**.

17.22.1 Parameters Related to cm_agent

log_dir

Parameter description: Specifies the directory where cm_agent logs are stored. The value can be an absolute path, or relative to the CM Agent data directory.

Value range: a string Any modification of this parameter takes effect only after cm_agent is restarted. For details about how to modify this parameter, see **Table 10-2**.

Default value: **"log"**, indicating that CM Agent logs are generated in the CM Agent data directory.

log_file_size

Parameter description: Specifies the size of a log file. If a log file exceeds the specified size, a new one is created to record log information.

Value range: an integer ranging from 0 to 2047. The unit is MB. For details about how to modify this parameter, see **Table 10-2**.

Default value: **16MB**

log_min_messages

Parameter description: Specifies which message levels are written to the cm_agent log. Each level covers all the levels following it. The lower the level is, the fewer messages will be written into the log.

Value range: enumerated type. Valid values are **debug5**, **debug1**, **log**, **warning**, **error**, and **fatal**. For details about how to modify this parameter, see **Table 10-2**.

Default value: **warning**

incremental_build

Parameter description: Specifies whether a standby DN is incrementally built. If this parameter is enabled, a standby DN is incrementally built.

Value range: Boolean. The value can be **on** or **off**. For details about how to modify this parameter, see **Table 10-2**.

Default value: **on**

alarm_component

Parameter description: Specifies the location of the alarm component that processes alarms.

Value range: a string For details about how to modify this parameter, see **Table 10-2**.

- If **--alarm-type** in the **gs_preinstall** script is set to **5**, no third-party component is connected and alarms are written into the **system_alarm** log. In this case, the value of **alarm_component** is **/opt/huawei/snas/bin/snas_cm_cmd**.
- If **--alarm-type** in the **gs_preinstall** script is set to **1**, a third-party component is connected. In this case, the value of **alarm_component** is the absolute path of the executable program of the third-party component.

Default value: **/opt/huawei/snas/bin/snas_cm_cmd**

alarm_report_interval

Parameter description: Specifies the interval at which an alarm is reported. For details about how to modify this parameter, see **Table 10-2**.

Value range: a non-negative integer (unit: s)

Default value: **1**

alarm_report_max_count

Parameter description: Specifies the maximum number of times an alarm is reported. For details about how to modify this parameter, see **Table 10-2**.

Value range: a non-negative integer

Default value: **1**

agent_report_interval

Parameter description: Specifies the interval at which cm_agent reports the instance status.

Value range: an integer. The unit is s. For details about how to modify this parameter, see **Table 10-2**.

Default value: **1**

agent_phony_dead_check_interval

Parameter description: Specifies the interval at which cm_agent checks whether the DN process is suspended.

Value range: an integer. The unit is s. For details about how to modify this parameter, see **Table 10-2**.

Default value: 10

agent_check_interval

Parameter description: Specifies the interval at which cm_agent queries for the status of instances, such as the DNs.

Value range: an integer. The unit is s. For details about how to modify this parameter, see **Table 10-2**.

Default value: 2

agent_heartbeat_timeout

Parameter description: Specifies the heartbeat timeout interval for CM Agent to connect to CM Server.

Value range: an integer ranging from 2 to $2^{31} - 1$. The unit is second. For details about how to modify this parameter, see **Table 10-2**.

Default value: **8**

agent_connect_timeout

Parameter description: Specifies the time to wait before the attempt of cm_agent to connect to cm_server times out.

Value range: an integer. The unit is s. For details about how to modify this parameter, see **Table 10-2**.

Default value: **1**

agent_connect_retries

Parameter description: Specifies the number of times cm_agent tries to connect to the cm_server.

Value range: an integer. For details about how to modify this parameter, see **Table 10-2**.

Default value: **15**

agent_kill_instance_timeout

Parameter description: Specifies the interval from the time when cm_agent fails to connect to the primary cm_server to the time when cm_agent kills all instances on the node.

Value range: an integer. For details about how to modify this parameter, see **Table 10-2**.

Default value: **0**, indicating that the operation of killing all instances on the node is not initiated.

security_mode

Parameter description: Specifies whether DNs are started in secure mode. If this parameter is set to **on**, DNs are started in secure mode. Otherwise, DNs are started in non-secure mode.

Value range: Boolean. The value can be **on** or **off**. For details about how to modify this parameter, see **Table 10-2**.

Default value: **off**

upgrade_from

Parameter description: Specifies the internal version number of the database before an in-place upgrade. Do not modify the value of this parameter.

Value range: a non-negative integer For details about how to modify this parameter, see **Table 10-2**.

Default value: **0**

process_cpu_affinity

Parameter description: Specifies whether to bind a primary DN process to a CPU core before starting the process. If this parameter is set to **0**, core binding will not be performed. If it is set to another value, core binding will be performed, and the number of physical CPU cores is 2^n. Restart the database and cm_agent for any modification of this parameter to take effect. Only ARM is supported. For details about how to modify this parameter, see **Table 10-2**.

Value range: an integer ranging from 0 to 2

Default value: **0**

log_threshold_check_interval

Parameter description: Specifies the interval for compressing and clearing logs. Logs are compressed and cleared every 1800 seconds.

Value range: an integer ranging from 0 to 2147483647. The unit is s. For details about how to modify this parameter, see **Table 10-2**.

Default value: **1800**

dilatation_shard_count_for_disk_capacity_alarm

Parameter description: Specifies the number of shards to be added in the scale-out scenario. This parameter is used to calculate the threshold for reporting a disk capacity alarm.

NOTE

The parameter value must be the same as the actual number of shards to be added.

Value range: an integer ranging from 0 to 2^{32} – 1. If this parameter is set to **0**, the disk scale-out alarm is not reported. If this parameter is set to a value greater than **0**, the disk scale-out alarm is reported and the threshold is calculated based on the number of shards specified by this parameter. For details about how to modify this parameter, see **Table 10-2**.

Default value: **1**

log_max_size

Parameter description: Specifies the maximum size of a log file.

Value range: an integer ranging from 0 to 2147483647. The unit is MB. For details about how to modify this parameter, see **Table 10-2**.

Default value: **10240**

log_max_count

Parameter description: Specifies the maximum number of logs that can be stored on hard disks.

Value range: an integer ranging from 0 to 10000. For details about how to modify this parameter, see **Table 10-2**.

Default value: **10000**

log_saved_days

Parameter description: Specifies the number of days for storing logs.

Value range: an integer ranging from 0 to 1000. The unit is day. For details about how to modify this parameter, see **Table 10-2**.

Default value: **90**

enable_log_compress

Parameter description: Specifies whether to enable log compression.

Value range: Boolean For details about how to modify this parameter, see **Table 10-2**.

- **on** indicates that log compression is enabled.
- **off** indicates that log compression is disabled.

Default value: **on**

agent_backup_open

Parameter description: Specifies the DR database instance settings. After this parameter is enabled, CM runs in DR database instance mode.

Value range: 0 or **1**. You need to restart **cm_agent** for the modification to take effect. For details about how to modify this parameter, see **Table 10-1**.

- **0**: disabled.
- **1**: enabled.

Default value: **0**

enable_xc_maintenance_mode

Parameter description: Specifies whether the **pgxc_node** system catalog can be modified when the database instance is in read-only mode.

Value range: Boolean Any modification of this parameter takes effect only after cm_agent is restarted. For details about how to modify this parameter, see **Table 10-1**.

- **on** indicates that the **pgxc_node** system catalog can be modified.
- **off** indicates that the **pgxc_node** system catalog cannot be modified.

Default value: **on**

unix_socket_directory

Parameter description: Specifies the directory location of the Unix socket.

Value range: a string For details about how to modify this parameter, see **Table 10-1**.

Default value: ''

enable_dcf

Parameter description: Specifies the status of the DCF mode.

Value range: Boolean Any modification of this parameter takes effect only after cm_agent is restarted. For details about how to modify this parameter, see **Table 10-1**.

- **0**: disabled.
- **1**: enabled.

Default value: **off**

disaster_recovery_type

Parameter description: Specifies the type of the DR relationship between the primary and standby database instances.

Value range: an integer ranging from 0 to 2 For details about how to modify this parameter, see **Table 10-1**.

- **0** indicates that no DR relationship is established.
- **1** indicates that the OBS DR relationship is established.
- **2** indicates that the streaming DR relationship is established.

Default value: **0**

17.22.2 Parameters Related to cm_server

log_dir

Parameter description: Specifies the directory where cm_server logs are stored. The value can be an absolute path, or relative to the CM Server data directory.

Value range: a string You need to restart cm_server for the modification to take effect. For details about how to modify this parameter, see **Table 10-2**.

Default value: **"log"**, indicating that CM Server logs are generated in the CM Server data directory.

log_file_size

Parameter description: Specifies the size of a log file. If a log file exceeds the specified size, a new one is created to record log information.

Value range: an integer ranging from 0 to 2047. The unit is MB. For details about how to modify this parameter, see **Table 10-2**.

Default value: **16MB**

log_min_messages

Parameter description: Specifies which message levels are written to the cm_server log. Each level covers all the levels following it. The lower the level is, the fewer messages will be written into the log.

Value range: enumerated type. Valid values are **debug5**, **debug1**, **log**, **warning**, **error**, and **fatal**. For details about how to modify this parameter, see **Table 10-2**.

Default value: **warning**

thread_count

Parameter description: Specifies the number of threads in the cm_server thread pool.

Value range: an integer ranging from 2 to 1000. You need to restart cm_server for the modification to take effect. For details about how to modify this parameter, see **Table 10-2**.

Default value: **1000**

alarm_component

Parameter description: Specifies the location of the alarm component that processes alarms.

Value range: a string For details about how to modify this parameter, see **Table 10-2**.

- If **--alarm-type** in the **gs_preinstall** script is set to **5**, no third-party component is connected and alarms are written into the **system_alarm** log. In this case, the value of **alarm_component** is **/opt/huawei/snas/bin/snas_cm_cmd**.
- If **--alarm-type** in the **gs_preinstall** script is set to **1**, a third-party component is connected. In this case, the value of **alarm_component** is the absolute path of the executable program of the third-party component.

Default value: **/opt/huawei/snas/bin/snas_cm_cmd**

instance_failover_delay_timeout

Parameter description: Specifies the delay in cm_server failover after the primary cm_server breakdown is detected.

Value range: an integer. The unit is s. For details about how to modify this parameter, see **Table 10-2**.

Default value: **0**

instance_heartbeat_timeout

Parameter description: Specifies the time to wait before the instance heartbeat times out.

Value range: an integer. The unit is s. For details about how to modify this parameter, see **Table 10-2**.

Default value: **6**

cmserver_ha_connect_timeout

Parameter description: Specifies the time to wait before the connection between the primary and standby cm_servers times out.

Value range: an integer. The unit is s. For details about how to modify this parameter, see **Table 10-2**.

Default value: 2

cmserver_ha_heartbeat_timeout

Parameter description: Specifies the time to wait before the heartbeat between the primary and standby cm_servers times out.

Value range: an integer. The unit is s. For details about how to modify this parameter, see **Table 10-2**.

Default value: **6**

phony_dead_effective_time

Parameter description: Specifies the maximum number of times DN processes are detected as zombie. If the number of times a process is detected as zombie is greater than the specified value, the process is considered to be a zombie process and will be restarted.

Value range: an integer. For details about how to modify this parameter, see **Table 10-2**.

Default value: **5**

enable_transaction_read_only

Parameter description: Specifies whether the database is in read-only mode.

Value range: Boolean values **on**, **off**, **true**, **false**, **yes**, **no**, **1**, and **0** For details about how to modify this parameter, see **Table 10-2**.

Default value: **on**

datastorage_threshold_check_interval

Parameter description: Specifies the interval for checking the disk usage. The system checks the disk usage at the interval specified by the user.

Value range: an integer. The unit is s. For details about how to modify this parameter, see **Table 10-2**.

Default value: **10**

datastorage_threshold_value_check

Parameter description: Specifies the usage threshold of a read-only disk in a database. When the disk usage of the data directory exceeds the specified value, the database is automatically set to read-only mode.

Value range: an integer ranging from 1 to 99, in percentage. For details about how to modify this parameter, see **Table 10-2**.

Default value: **85**

max_datastorage_threshold_check

Parameter description: Specifies the maximum interval for checking the disk usage. After you modify the **enable_transaction_read_only** parameter, the system automatically checks whether the disk usage reaches the threshold at the specified interval.

Value range: an integer. The unit is s. For details about how to modify this parameter, see **Table 10-2**.

Default value: **43200**

cmserver_ha_status_interval

Parameter description: Specifies the interval between synchronizations of primary and standby CM Server status.

Value range: an integer. The unit is s. For details about how to modify this parameter, see **Table 10-2**.

Default value: **1**

cmserver_self_vote_timeout

Parameter description: Specifies the time to wait before the CM Server self-vote times out.

Value range: an integer. The unit is s. For details about how to modify this parameter, see **Table 10-2**.

Default value: **6**

alarm_report_interval

Parameter description: Specifies the interval at which an alarm is reported.

Value range: a non-negative integer. The unit is s. For details about how to modify this parameter, see **Table 10-2**.

Default value: **3**

alarm_report_max_count

Parameter description: Specifies the maximum number of times an alarm is reported.

Value range: a non-negative integer For details about how to modify this parameter, see **Table 10-2**.

Default value: **1**

enable_az_auto_switchover

Parameter description: Specifies whether to enable automatic AZ switchover. If it is set to **1**, cm_server automatically switches over services among AZs. Otherwise, when a DN is faulty, services will not be automatically switched to another AZ even if the current AZ is unavailable. You can run the switchover command to manually switch services to another AZ.

Value range: a non-negative integer. The value **0** indicates that automatic AZ switchover is disabled, and the value **1** indicates that automatic AZ switchover is enabled. For details about how to modify this parameter, see **Table 10-2**.

Default value: 1

instance_keep_heartbeat_timeout

Parameter description: The cm_agent periodically checks the instance status and reports the status to the cm_server. If the instance status cannot be detected for a long time and the accumulated number of times exceeds the value of this parameter, the cm_server delivers a command to the cm_agent to restart the instance.

Value range: an integer. The unit is s. For details about how to modify this parameter, see **Table 10-2**.

Default value: 40

az_switchover_threshold

Parameter description: If the failure rate of a DN shard in an AZ (Number of faulty DN shards/Total number of DN shards x 100%) exceeds the specified value, automatic AZ switchover is triggered.

Value range: an integer ranging from 0 to 100 For details about how to modify this parameter, see **Table 10-2**.

Default value: 100

az_check_and_arbitrate_interval

Parameter description: Specifies the interval for checking the AZ status. If the status of an AZ is abnormal, automatic AZ switchover is triggered.

Value range: an integer. The unit is s. For details about how to modify this parameter, see **Table 10-2**.

Default value: 2

az_connect_check_interval

Parameter description: Specifies the interval at which the network connection between AZs is checked.

Value range: an integer. The unit is s. For details about how to modify this parameter, see **Table 10-2**.

Default value: 60

az_connect_check_delay_time

Parameter description: Specifies the delay between two retries to check the network connection between AZs.

Value range: an integer. The unit is s. For details about how to modify this parameter, see **Table 10-2**.

Default value: **150**

cmserver_demote_delay_on_etcd_fault

Parameter description: Specifies the interval at which cm_server switches from the primary state to the standby state due to unhealthy etcd.

Value range: an integer. The unit is s. For details about how to modify this parameter, see **Table 10-2**.

Default value: **8**

instance_phony_dead_restart_interval

Parameter description: Specifies the interval at which the cm_agent process restarts and kills a zombie DN instance. The interval between two consecutive kill operations cannot be less than the value of this parameter. Otherwise, the cm_agent process does not deliver commands.

Value range: an integer. The unit is s. The minimum value that takes effect is **1800**. If this parameter is set to a value less than **1800**, the value **1800** takes effect. For details about how to modify this parameter, see **Table 10-2**.

Default value: 21600

cm_auth_method

Parameter description: Specifies the port authentication mode of the CM. **trust** indicates that port authentication is not configured. **gss** indicates that Kerberos port authentication is used. Note that you can change the value to **gss** only after the Kerberos server and client are successfully installed. Otherwise, the CM cannot communicate properly, affecting the database status.

Value range: **gss** or **trust**. For details about how to modify this parameter, see **Table 10-2**.

Default value: **trust**

cm_krb_server_keyfile

Parameter description: Specifies the location of the key file on the Kerberos server. The value must be an absolute path. The file is usually stored in the *${GAUSSHOME}***/kerberos** directory and ends with keytab. The file name is the same as the name of the user who runs the database. This parameter is used together with **cm_auth_method**. If the **cm_auth_method** parameter is changed to **gss**, **cm_krb_server_keyfile** must also be configured as the correct path. Otherwise, the database status will be affected.

Value range: a string. For details about how to modify the parameter, see **Table 10-2**.

Default value: *${GAUSSHOME}*/**kerberos**/*{UserName}*.**keytab**. The default value cannot take effect and is used only as a prompt.

cm_server_arbitrate_delay_base_time_out

Parameter description: Specifies the basic delay duration for cm_server quorum. If cm_server is disconnected, the quorum starts to be timed. If the disconnection duration exceeds the quorum delay duration, a new cm_server will be selected. The quorum delay duration is determined by the basic delay duration, the node index (server ID), and the incremental delay duration. The formula is as follows: Quorum delay duration = Basic delay duration + Node index x Incremental delay duration

Value range: an integer. The unit is s. The index should be larger than 0. For details about how to modify this parameter, see **Table 10-2**.

Default value: 10

cm_server_arbitrate_delay_incrememtal_time_out

Parameter description: Specifies the incremental delay duration for cm_server quorum. If cm_server is disconnected, the quorum starts to be timed. If the disconnection duration exceeds the quorum delay duration, a new cm_server will be selected. The quorum delay duration is determined by the basic delay duration, the node index (server ID), and the incremental delay duration. The formula is as follows: Quorum delay duration = Basic delay duration + Node index x Incremental delay duration

Value range: an integer. The unit is s. The index should be larger than 0. For details about how to modify this parameter, see **Table 10-2**.

Default value: **3**

force_promote

Parameter description: Specifies whether cm_server enables the forcible startup logic (that is, when the cluster status is unknown, ensure that the basic functions of the cluster are available at the cost of data loss). The value **0** indicates that the function is disabled, and the value **1** indicates that the function is enabled. This parameter applies to DNs.

Value range: 0 or **1**. For details about how to modify this parameter, see **Table 10-2**.

Default value: **0**

switch_rto

Parameter description: Specifies the delay for the forcible startup of cm_server. When **force_promote** is set to **1** and a shard in the database does not have primary cm_server, the system starts timing. After the delay, the forcible startup logic starts to be executed.

Value range: an integer ranging from 60 to 2147483647. The unit is s. For details about how to modify this parameter, see **Table 10-2**.

Default value: **600**

backup_open

Parameter description: Specifies the DR database instance settings. After this parameter is enabled, CM runs in DR database instance mode.

Value range: 0 or **1**. You need to restart cm_server for the modification to take effect. This parameter cannot be enabled for a non-DR database instance. For details about how to modify this parameter, see **Table 10-2**.

- **0**: disabled.
- **1**: enabled.

Default value: **0**

enable_dcf

Parameter description: Specifies the status of the DCF mode.

Value range: Boolean You need to restart cm_server for the modification to take effect. For details about how to modify this parameter, see **Table 10-2**.

- **0**: disabled.
- **1**: enabled.

Default value: **off**

install_type

Parameter description: Specifies the settings related to the DR database instance. It is used to distinguish whether the database instance is based on Dorado.

Value range: an integer ranging from 0 to 2 You need to restart cm_server for the modification to take effect. This parameter cannot be enabled for a non-DR database instance. For details about how to modify this parameter, see **Table 10-2**.

- **0** indicates the database instance for which the DR relationship is not established.
- **1** indicates a Dorado-based database instance.
- **2** indicates a streaming-based database instance.

Default value: **0**

enable_ssl

Parameter description: Specifies whether to enable SSL.

Value range: Boolean After this function is enabled, the SSL certificate is used to encrypt communication. Any modification of this parameter takes effect only after a restart. For details about how to modify this parameter, see **Table 10-2**.

- **on** indicates that SSL is enabled.
- **off** indicates that SSL is disabled.
- **Default value**: **off**

NOTICE

To ensure security, you are advised not to disable it. After this function is disabled, the CM **does not** use encrypted communication and all information is transmitted in plaintext, which may bring security risks such as eavesdropping, tampering, and spoofing.

ssl_cert_expire_alert_threshold

Parameter description: Specifies the SSL certificate expiration alarm time.

Value range: an integer. The unit is day. If the certificate expiration time is less than the value of this parameter, an alarm indicating that the certificate is about to expire is reported. Any modification of this parameter takes effect only after a restart. For details about how to modify this parameter, see **Table 10-2**.

Default value: **90**

ssl_cert_expire_check_interval

Parameter description: Specifies the period for checking whether the SSL certificate expires.

Value range: an integer. The unit is s. Any modification of this parameter takes effect only after a restart. For details about how to modify this parameter, see **Table 10-2**.

Default value: **86400**

delay_arbitrate_timeout

Parameter description: Specifies the waiting time for a node in the same AZ as the primary DN to be promoted to primary after redo replay.

Value range: an integer, in the range [0,21474836] (unit: second). For details about how to modify this parameter, see **Table 10-2**.

Default value: **0**

ddb_type

Parameter description: Specifies whether to switch between ETCD and DCC modes.

Value range: an integer. **0**: ETCD; **1**: DCC. You need to restart cm_server for the modification to take effect. For details about how to modify this parameter, see **Table 10-2**.

Default value: **0**

ddb_log_level

Parameter description: Sets the DDB log level.

To disable the log function, set this parameter to **NONE**, which cannot be used together with the following log levels:

To enable the log function, set this parameter to one or a combination of the following log levels: **RUN_ERR|RUN_WAR|RUN_INF|DEBUG_ERR|DEBUG_WAR|DEBUG_INF|TRACE|PROFILE|OPER**. If two or more log levels are used together, separate them with vertical bars (|). The log level cannot be set to an empty string.

Value range: a string containing one or a combination of the following log levels: **RUN_ERR|RUN_WAR|RUN_INF|DEBUG_ERR|DEBUG_WAR|DEBUG_INF|TRACE|PROFILE|OPER**. For details about how to modify this parameter, see **Table 10-2**.

Default value: **RUN_ERR|RUN_WAR|DEBUG_ERR|OPER|RUN_INF|PROFILE**

ddb_log_backup_file_count

Parameter description: Specifies the maximum number of log files that can be saved.

Value range: an integer ranging from 1 to 100. For details about how to modify this parameter, see **Table 10-2**.

Default value: **10**

ddb_max_log_file_size

Parameter description: Specifies the maximum number of bytes in a log.

Value range: a string, in the range [1M,1000M]. For details about how to modify this parameter, see **Table 10-2**.

Default value: **10M**

ddb_log_suppress_enable

Parameter description: Specifies whether to enable the log suppression function.

Value range: an integer. **0**: disabled; **1**: enabled. For details about how to modify this parameter, see **Table 10-2**.

Default value: **1**

ddb_election_timeout

Parameter description: Specifies the DCC election timeout period.

Value range: an integer, in the range [1,600], in seconds. For details about how to modify this parameter, see **Table 10-2**.

Default value: **3**

17.23 Upgrade Parameters

IsInplaceUpgrade

Parameter description: Specifies whether an upgrade is ongoing. This parameter cannot be modified by users. Only the sysadmin user can access this parameter.

This parameter is a SUSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates an upgrade is ongoing.
- **off** indicates no upgrade is ongoing.

Default value: **off**

inplace_upgrade_next_system_object_oids

Parameter description: Indicates the OID of a new system object during the in-place upgrade. The value of this parameter cannot be changed.

This parameter is a SUSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string

Default value: empty

upgrade_mode

Parameter description: Specifies the upgrade mode.

This parameter is a fixed INTERNAL parameter and cannot be modified.

Value range: an integer ranging from 0 to *INT_MAX*

- **0** indicates that no upgrade is ongoing.
- **1** indicates that a local upgrade is ongoing.
- **2** indicates that a grayscale upgrade is ongoing.

Default value: **0**

NOTE

Special case: When the gray upgrade is used, if the major version upgrade policy is selected, that is, the upgrade script needs to be executed and the binary package needs to be replaced, the value of **upgrade_mode** is set to **2**;iIf the minor version upgrade policy is selected, that is, only the binary package needs to be replaced, the value of **upgrade_mode** is not set to **2**.

17.24 Miscellaneous Parameters

enable_default_ustore_table

Parameter description: Specifies whether to enable the UStore storage engine by default. If this parameter is set to **on**, all created tables are UStore tables.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**. Note that the **track_counts** and **track_activities** parameters must be enabled when the Ustore table is used. Otherwise, space expansion may occur.

Value range: [off,on]

Default value: **off**

enable_ustore

Parameter description: Specifies whether to enable the Ustore storage engine. If this parameter is set to **on**, Ustore tables can be created. Note that the **track_counts** and **track_activities** parameters must be enabled when the Ustore table is used. Otherwise, space expansion may occur.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: [off,on]

Default value: **on**

reserve_space_for_nullable_atts

Parameter description: Specifies whether to reserve space for the nullable attribute of an Ustore table. If this parameter is set to **on**, space is reserved for the nullable attribute of the Ustore table by default.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: [off,on]

Default value: **on**

ustore_attr

Parameter description: Specifies the UStore test parameters.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

You can set **enable_ustore_partial_seqscan** (copy selective columns only during sequential scanning in the ustore table), **enable_candidate_buf_usage_count** (whether dirty pages are evicted and added to the use count weight), and **ustats_tracker_naptime** (time for reloading the statistics file). **umax_search_length_for_prune** (number of blocks to be pruned before table extension) and **ustore_unit_test** (starting the UStore white box test). The setting method is ustore_att ='**enable_ustore_partial_seqscan**=on'.

Value range: a string

server_version

Parameter description: Specifies the server version number.

This parameter is a fixed parameter of the INTERNAL type. It can be viewed but cannot be modified. This parameter is inherited from the PostgreSQL kernel, indicating that the current database kernel is compatible with the server_version version corresponding to PostgreSQL. This parameter is reserved to ensure the ecosystem compatibility of the northbound external tool interface (query when the tool is connected). This parameter is not recommended. To query the server version, use the opengauss_version() function.

Value range: a string

Default value: **9.2.4**

server_version_num

Parameter description: Specifies the server version number.

This parameter is a fixed parameter of the INTERNAL type. It can be viewed but cannot be modified. This parameter is inherited from the PostgreSQL kernel, indicating that the current database kernel is compatible with the server_version_num version corresponding to PostgreSQL. This parameter is reserved to ensure the ecosystem compatibility of the northbound external tool API. (query when the tool is connected).

Value range: an integer

Default value: **90204**

block_size

Parameter description: Specifies the block size of the current database.

This parameter is a fixed INTERNAL parameter and cannot be modified.

Value: **8192**

Default value: **8192**

segment_size

Parameter description: Specifies the segment file size of the current database.

This parameter is a fixed INTERNAL parameter and cannot be modified.

Unit: 8 KB

Default value: 131072, that is, 1 GB

max_index_keys

Parameter description: Specifies the maximum number of index keys supported by the current database.

This parameter is a fixed INTERNAL parameter and cannot be modified.

Default value: **32**

integer_datetimes

Parameter description: Specifies whether the date and time are in the 64-bit integer format.

This parameter is a fixed INTERNAL parameter and cannot be modified.

Value range: Boolean

- **on** indicates that the 64-bit integer format is used.
- **off** indicates that the 64-bit integer format is not used.

Default value: **on**

lc_collate

Parameter description: Specifies the locale in which sorting of textual data is done.

This parameter is a fixed INTERNAL parameter and cannot be modified.

Default value: Determined by the configuration set during the database installation and deployment.

lc_ctype

Parameter description: Specifies the locale that determines character classifications. For example, it specifies what a letter and its upper-case equivalent are.

This parameter is a fixed INTERNAL parameter and cannot be modified.

Default value: Determined by the configuration set during the database installation and deployment.

max_identifier_length

Parameter description: Specifies the maximum identifier length.

This parameter is a fixed INTERNAL parameter and cannot be modified.

Value range: an integer

Default value: **63**

server_encoding

Parameter description: Specifies the database encoding (character set).

By default, gs_initdb will initialize the setting of this parameter based on the current system environment. You can also run the **locale** command to check the current configuration environment.

This parameter is a fixed INTERNAL parameter and cannot be modified.

Default value: determined by the current system environment when the database is created.

enable_upgrade_merge_lock_mode

Parameter description: If this parameter is set to **on**, the delta merge operation internally increases the lock level, and errors can be prevented when update and delete operations are performed at the same time.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- If this parameter is set to **on**, the delta merge operation internally increases the lock level. In this way, when the **DELTAMERGE** operation is concurrently performed with the **UPDATE** or **DELETE** operation, one operation can be performed only after the previous one is complete.
- If this parameter is set to **off** and the **DELTAMERGE** operation is concurrently performed with the **UPDATE** or **DELETE** operation to the data in a row in the delta table of the table, errors will be reported during the later operation, and the operation will stop.

Default value: **off**

transparent_encrypted_string

Parameter description: Specifies a sample string that is transparently encrypted. Its value is generated by encrypting **TRANS_ENCRYPT_SAMPLE_STRING** using a database secret key. The ciphertext is used to check whether the DEK obtained during secondary startup is correct. If it is incorrect, database nodes will not be started. This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**. This parameter applies only to the DWS scenario in the current version.

Value range: a string. An empty string indicates that the entire database is a not encrypted.

Default value: empty

NOTE

Do not set this parameter manually. Otherwise, the database may become faulty.

transparent_encrypt_kms_url

Parameter description: Specifies the URL for obtaining the database secret key to be transparently encrypted. It must contain only the characters specified in RFC3986, and the maximum length is 2047 bytes. The format is **kms://***Protocol***@***KMS host name 1***;***KMS host name 2***:***KMS port number***/kms**, for example, **kms://https@linux175:29800/**. This parameter applies only to the DWS scenario in the current version.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string

Default value: empty

transparent_encrypt_kms_region

Parameter description: Specifies the deployment region of the entire database. It must contain only the characters specified in RFC3986, and the maximum length is 2047 bytes. This parameter applies only to the DWS scenario in the current version.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string

Default value: empty

basebackup_timeout

Parameter description: Specifies the timeout interval for a connection that has no read or write operations after a backup transfer is complete.

When the gs_basebackup tool is used for transmission and a high compression rate is specified, the transmission of the tablespace may time out (the client needs to compress the transmitted data).

Value range: an integer ranging from 0 to *INT_MAX*. The unit is s. **0** indicates that archiving timeout is disabled.

Default value: 600s

datanode_heartbeat_interval

Parameter description: Specifies the interval at which heartbeat messages are sent between heartbeat threads. You are advised to set this parameter to a value no more than wal_receiver_timeout/2.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 1000 to 60000. The unit is ms.

Default value: **1s**

max_concurrent_autonomous_transactions

Parameter description: Specifies the maximum number of autonomous transaction connections, that is, the maximum number of concurrent autonomous transactions executed at the same time. If this parameter is set to **0**, autonomous transactions cannot be executed.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: 0 to1024

Default value: 10

cluster_run_mode

Parameter description: Specifies whether a DN belongs to the primary or standby database instance in the dual-database instance DR scenario. For a single database instance, use the default primary database instance.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: enumerated values

- **cluster_primary** indicates the primary database instance.
- **cluster_standby** indicates the standby database instance.

Default value: **cluster_primary**

acceleration_with_compute_pool

Parameter description: Specifies whether to use the computing resource pool for acceleration when an OBS is queried. (Due to specification changes, the current version no longer supports the current feature. Do not use this feature.)

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the query covering the OBS is accelerated based on the cost when the computing resource pool is available.
- **off** indicates that no query is accelerated using the computing resource pool.

Default value: **off**

dfs_partition_directory_length

Parameter description: Specifies the maximum directory name length for the partition directory of a table partitioned by VALUE in the HDFS.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: 92 to 7999

Default value: **512**

max_resource_package

Parameter description: Specifies the maximum number of threads that each DN can run concurrently on an acceleration database instance on the cloud.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: 0 to 2147483647

Default value: **0**

17.25 Wait Events

enable_instr_track_wait

Parameter description: Specifies whether to enable real-time collection of wait event information.

In the x86-based centralized deployment scenario, the hardware configuration specifications are 32-core CPU and 256 GB memory. When the Benchmark SQL 5.0 tool is used to test performance, the performance fluctuates by about 1.4% by enabling or disabling this parameter.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the function of collecting wait event information is enabled.
- **off** indicates that the function of collecting wait event information is disabled.

Default value: **on**

17.26 Query

instr_unique_sql_count

Parameter description: Specifies the maximum number of Unique SQL records to be collected. The value **0** indicates that the function of collecting Unique SQL information is disabled.

If the value is changed from a larger one to a smaller one, the original data in the system will be cleared and re-collected (the standby node does not support this function). There is no impact if the value is changed from a smaller one to a larger one.

When the number of unique SQL records generated in the system is greater than the value of **instr_unique_sql_count**, the extra unique SQL records are not collected.

In the x86-based centralized deployment scenario, the hardware configuration specifications are 32-core CPU and 256 GB memory. When the Benchmark SQL 5.0 tool is used to test performance, the performance fluctuates by about 3% by enabling or disabling this parameter.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 2147483647

Default value: **200000**

instr_unique_sql_track_type

Parameter description: Specifies which SQL statements are recorded in Unique SQL.

This parameter is an INTERNAL parameter. Set it based on instructions provided in **Table 10-1**.

Value range: enumerated values

top: Only top-level SQL statements are recorded.

Default value: **top**

enable_instr_rt_percentile

Parameter description: Specifies whether to enable the function of calculating the response time of 80% and 95% SQL statements in the system.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the function of calculating the response time of 80% and 95% SQL statements is enabled.
- **off** indicates that the function of calculating the response time of 80% and 95% SQL statements is disabled.

Default value: **on**

percentile

Parameter description: Specifies the percentage of SQL statements whose response time is to be calculated by the background calculation thread.

This parameter is an INTERNAL parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string

Default value: **80, 95**

instr_rt_percentile_interval

Parameter description: Specifies the interval at which the background calculation thread calculates the SQL response time.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 3600. The unit is s.

Default value: **10s**

enable_instr_cpu_timer

Parameter description: Specifies whether to capture the CPU time consumed during SQL statement execution.

In the x86-based centralized deployment scenario, the hardware configuration specifications are 32-core CPU and 256 GB memory. When the Benchmark SQL 5.0 tool is used to test performance, the performance fluctuates by about 3.5% by enabling or disabling this parameter.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates the CPU time consumed during SQL statement execution is captured.
- **off** indicates the CPU time consumed during SQL statement execution is not captured.

Default value: **on**

enable_stmt_track

Parameter description: Specifies whether to enable the full/slow SQL statement feature.

In the x86-based centralized deployment scenario, the hardware configuration specifications are 32-core CPU and 256 GB memory. When the Benchmark SQL 5.0 tool is used to test performance, the performance fluctuates by about 1.2% by enabling or disabling this parameter.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- on: Full/Slow SQL capture is enabled.
- off: Full /Slow SQL capture is disabled.

Default value: **on**

track_stmt_parameter

Parameter description: After **track_stmt_parameter** is enabled, the executed statements recorded in **statement_history** are not normalized. The complete SQL statement information can be displayed to help the database administrator locate problems. For a simple query, the complete statement information is displayed. For a PBE statement, the complete statement information and information about each variable value are displayed. The format is query string; parameters: $1=value1,$2=value2, ... This parameter is used to display complete SQL information and is not controlled by the **track_activity_query_size** parameter.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on**: The function of displaying complete SQL statement information is enabled.
- **off**: The function of displaying complete SQL statement information is disabled.

Default value: **off**

track_stmt_session_slot

Parameter description: Specifies the maximum number of full/slow SQL statements that can be cached in a session. If the number of full/slow SQL statements exceeds this value, new statements will not be traced until the flush thread flushes the cached statements to the disk to reserve idle space.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 2147483647

Default value: **1000**

track_stmt_details_size

Parameter description: Specifies the maximum size (in bytes) of execution events that can be collected by a single statement.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 100000000

Default value: **4096**

track_stmt_retention_time

Parameter description: Specifies the retention period of full/slow SQL statement records. This parameter is a combination of parameters. This parameter is read every 60 seconds and records exceeding the retention period are deleted. Only the sysadmin user can access this parameter.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string

This parameter consists of two parts in the format of 'full sql retention time, slow sql retention time'.

full sql retention time indicates the retention time of full SQL statements. The value ranges from 0 to 86400.

slow sql retention time indicates the retention time of slow SQL statements. The value ranges from 0 to 604800.

Default value: **3600,604800**

track_stmt_stat_level

Parameter description: Determines the level of statement execution tracing.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**. The value is case-insensitive.

Value range: a string

This parameter consists of two parts in the format of 'full sql stat level, slow sql stat level'.

The first part indicates the tracing level of full SQL statements. The value can be **OFF**, **L0**, **L1**, or **L2**.

The second part indicates the tracing level of slow SQL statements. The value can be **OFF**, **L0**, **L1**, or **L2**.

NOTE

If the full SQL tracing level is not **OFF**, the current SQL tracing level is the higher level (L2 > L1 > L0) of the full SQL and slow SQL statements. For details about the levels, see **Table 13-113**.

Default value: **OFF,L0**

enable_auto_clean_unique_sql

Parameter description: Specifies whether to enable the automatic elimination function of unique SQL statements when the number of unique SQL statements generated in the system is greater than or equal to the value of **instr_unique_sql_count**.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

Default value: **off**

CAUTION

Some snapshot information comes from unique SQL statements. Therefore, when automatic elimination is enabled, if the selected start snapshot and end snapshot exceed the elimination time, the WDR report cannot be generated.

asp_log_directory

Parameter description: Specifies the directory for storing ASP log files on the server when **asp_flush_mode** is set to **all** or **file**. The value can be an absolute path, or relative to the **data** directory. Only the sysadmin user can access this parameter.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

If the value of **asp_log_directory** in the configuration file is an invalid path, the database instance cannot be restarted.

NOTE

- Valid path: Users must have read and write permissions on the path.
- Invalid path: Users do not have read or write permissions on an invalid path.

Value range: a string

Default value: specified during installation

enable_slow_query_log (Discarded)

Parameter description: Specifies whether to write the slow query information to the log file. This parameter is discarded in this version.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on**: indicates that slow query information needs to be written into log files.
- **off**: indicates that slow query information does not need to be written into log files.

Default value: **on**

query_log_file (Discarded)

Parameter description: If **enable_slow_query_log** is set to **ON**, slow query records are written into log files. **query_log_file** specifies the name of a slow query log file on the server. Only the sysadmin user can access this parameter. Generally, log file names are generated in strftime mode. Therefore, the system time can be used to define log file names, which are implemented using the escape character %. This function has been discarded in this version.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

You are advised to use %-escapes to specify the log file names for efficient management of log files.

Value range: a string

Default value: slow_query_log-%Y-%m-%d_%H%M%S.log

query_log_directory (Discarded)

Parameter description: Specifies the directory for storing low query log files when **enable_slow_query_log** is set to **on**. Only the sysadmin user can access this parameter. It can be an absolute path or a relative path (relative to the data directory), which has been discarded in this version.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

NOTICE

If the value of **query_log_directory** in the configuration file is an invalid path, the database instance cannot be restarted.

NOTE

Valid path: Users have read and write permissions on the path.

Invalid path: You do not have read or write permission on the path.

Value range: a string

Default value: specified during installation

perf_directory

Parameter description: Specifies the directory of the output file of the performance view dotting task. Only the sysadmin user can access this parameter. The value can be an absolute path, or relative to the data directory.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

NOTE

- Valid path: Users must have read and write permissions on the path.
- Invalid path: Users do not have read or write permissions on an invalid path.

Value range: a string

Default value: specified during installation

unique_sql_retention_time

Parameter description: Specifies the memory cleanup interval for the unique SQL hash table. The default value is 30 minutes.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 1 to 3650

Default value: **30min**

17.27 System Performance Snapshot

enable_wdr_snapshot

Parameter description: Specifies whether to enable the database monitoring snapshot function.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the database monitoring snapshot function is enabled.
- **off** indicates that the database monitoring snapshot function is disabled.

Default value: **on**

wdr_snapshot_retention_days

Parameter description: Specifies the number of days for storing database monitoring snapshot data in the system. When the number of snapshots generated during database running exceeds the maximum number of snapshots that can be generated within the retention period, the system clears the snapshot data with the smallest **snapshot_id** at the interval specified by **wdr_snapshot_interval**.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 1 to 8

Default value: **8**

wdr_snapshot_query_timeout

Parameter description: Specifies the execution timeout for the SQL statements associated with database monitoring snapshot operations. If the SQL statement execution is not complete and a result is not returned within the specified time, the snapshot operation fails.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 100 to *INT_MAX*. The unit is s.

Default value: **100s**

wdr_snapshot_interval

Parameter description: Specifies the interval at which the backend thread Snapshot automatically performs snapshot operations on the database monitoring data.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 10 to 60. The unit is min.

Default value: **1h**

asp_flush_mode

Parameter description: Specifies the mode for the ASP to update data to the disk. The value can be **file** (default value), **table** (system catalog), or **all** (system catalog and file). Only the sysadmin user can access this parameter.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string, which can be **table**, **file**, or **all**

Default value: **table**

asp_flush_rate

Parameter description: When the number of samples reaches the value of **asp_sample_num**, the samples in the memory are updated to the disk based on a certain proportion. **asp_flush_rate** indicates the update proportion. If this parameter is set to **10**, it indicates that the update ratio is 10:1.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 1 to 10

Default value: **10**

asp_log_filename

Parameter description: Specifies the file name format when writing files using ASP. Only the sysadmin user can access this parameter.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string

Default value: **asp-%Y-%m-%d_%H%M%S.log**

asp_retention_days

Parameter description: Specifies the maximum number of days for reserving ASP samples when they are written to the system catalog.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 1 to 7

Default value: **2**

asp_sample_interval

Parameter description: Specifies the sampling interval.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 1 to 10. The unit is s.

Default value: **1s**

asp_sample_num

Parameter description: Specifies the maximum number of samples allowed in the LOCAL_ACTIVE_SESSION view. Only the sysadmin user can access this parameter.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 10 to 100000.

Default value: **100000**

enable_asp

Parameter description: Specifies whether to enable the active session profile function.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on**: The function is enabled.
- **off**: The function is disabled.

Default value: **on**

17.28 Security Configuration

elastic_search_ip_addr

Parameter description: Specifies the IP address of the Elasticsearch system.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string

Default value: **'https:127.0.0.1'**

enable_security_policy

Parameter description: Specifies whether the unified audit and dynamic data masking policies take effect.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

on: The security policy is enabled.

off: The security policy is disabled.

Default value: **off**

use_elastic_search

Parameter description: Specifies whether to send unified audit logs to Elasticsearch. If **enable_security_policy** and this parameter are enabled, unified audit logs are sent to Elasticsearch through HTTP or HTTPS (used by default). After this parameter is enabled, ensure that the Elasticsearch service corresponding to **elastic_search_ip_addr** can be properly connected. Otherwise, the process fails to be started.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

on: Unified audit logs are sent to Elasticsearch.

off: Unified audit logs are not sent to Elasticsearch.

Default value: **off**

is_sysadmin

Parameter description: Specifies whether the current user is an initial user.

This parameter is a fixed INTERNAL parameter and cannot be modified.

Value range: Boolean

on indicates that the user is an initial user.

off indicates that the user is not an initial user.

Default value: **off**

enable_tde

Parameter description: Specifies whether to enable the TDE function. Set this parameter to **on** before creating an encrypted table. If this parameter is set to **off**, new encrypted tables cannot be created. The created encrypted table is decrypted only when data is read and is not encrypted when the data is written.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

on: The TDE function is enabled.

off: The TDE function is disabled.

Default value: **off**

tde_cmk_id

Parameter description: Specifies the CMK ID of the database instance used by the TDE function. The ID is generated by KMS. The CMK of the database instance is used to encrypt the DEK. When the DEK needs to be decrypted, a request packet needs to be sent to KMS. The DEK ciphertext and the ID of the corresponding CMK are sent to KMS.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string

Default value: ""

17.29 Global Temporary Table

max_active_global_temporary_table

Parameter description: Specifies whether global temporary tables can be created. Currently, the Ustore engine does not support global temporary tables. The value of this parameter determines the memory reserved in the shared cache for hash tables required by global temporary tables. The total number of active global temporary tables in all sessions is not forcibly limited.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 1000000

- **0**: The global temporary table function is disabled.
- > 0: The global temporary table function is enabled.

Default value: **1000**

vacuum_gtt_defer_check_age

Parameter description: Checks the differences between the global temporary table relfrozenxid and the ordinary table after VACUUM is executed. WARNING is generated if the difference value exceeds the specified parameter value. Use the default value for this parameter.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 1000000

Default value: **10000**

enable_gtt_concurrent_truncate

Parameter description: Specifies whether to support concurrent execution of TRUNCATE TABLE and DML operations on global temporary tables and concurrent execution of TRUNCATE TABLE on global temporary tables.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on/true** indicates that the preceding operations can be performed concurrently.
- **off/false** indicates that the preceding operations cannot be performed concurrently.

Default value: **on**

17.30 HyperLogLog

hll_default_log2m

Parameter description: Specifies the number of buckets for HLL data. The number of buckets affects the precision of distinct values calculated by HLL. The more buckets there are, the smaller the deviation is. The deviation range is as follows: $[–1.04/2^{log2m*1/2}, +1.04/2^{log2m*1/2}]$

This parameter is a **USERSET** parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 10 to 16

Default value: **14**

hll_default_log2explicit

Parameter description: Specifies the default threshold for switching from the explicit mode to the sparse mode.

This parameter is a **USERSET** parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 12 The value **0** indicates that the explicit mode is skipped. The value 1 to 12 indicates that the mode is switched when the number of distinct values reaches $2^{hll\_default\_log2explicit}$.

Default value: **10**

hll_default_log2sparse

Parameter description: Specifies the default threshold for switching from the **sparse** mode to the **full** mode.

This parameter is a **USERSET** parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 14 The value **0** indicates that the explicit mode is skipped. The value 1 to 14 indicates that the mode is switched when the number of distinct values reaches $2^{hll\_default\_log2sparse}$.

Default value: **12**

hll_duplicate_check

Parameter description: Specifies whether duplicatecheck is enabled by default.

This parameter is a **USERSET** parameter. Set it based on instructions provided in **Table 10-1**.

Value range: **0** or **1** **0**: disabled; **1**: enabled

Default value: **0**

hll_default_regwidth (Discarded)

Parameter description: Specifies the number of bits in each bucket for HLL data. A larger value indicates more memory occupied by HLL. **hll_default_regwidth** and **hll_default_log2m** determine the maximum number of distinct values that can be calculated by HLL. Currently, **regwidth** is set to a fixed value and is no longer used.

This parameter is a **USERSET** parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 1 to 5

Default value: **5**

hll_default_expthresh (Discarded)

Parameter description: Specifies the default threshold for switching from the **explicit** mode to the **sparse** mode. Currently, the **hll_default_log2explicit** parameter is used to replace the similar function.

This parameter is a **USERSET** parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from –1 to 7 **–1** indicates the auto mode; **0** indicates that the **explicit** mode is skipped; a value from 1 to 7 indicates that the mode is switched when the number of distinct values reaches $2^{hll\_default\_expthresh}$.

Default value: **–1**

hll_default_sparseon (Discarded)

Parameter description: Specifies whether to enable the **sparse** mode by default. Currently, the **hll_default_log2sparse** parameter is used to replace the similar function. When **hll_default_log2sparse** is set to **0**, the **sparse** mode is disabled.

This parameter is a **USERSET** parameter. Set it based on instructions provided in **Table 10-1**.

Value range: **0** or **1** **0** indicates that the **sparse** mode is disabled by default. **1** indicates that the **sparse** mode is enabled by default.

Default value: **1**

hll_max_sparse (Discarded)

Parameter description: Specifies the size of **max_sparse**. Currently, the **hll_default_log2sparse** parameter is used to replace the similar function.

This parameter is a **USERSET** parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from –1 to 2147483647

Default value: **–1**

enable_compress_hll (Discarded)

Parameter description: Specifies whether to enable memory optimization for HLL. Currently, the HLL memory has been optimized, and this parameter is no longer used.

This parameter is a **USERSET** parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** or **true** indicates that memory optimization is enabled.
- **off** or **false** indicates that memory optimization is disabled.

Default value: **off**

17.31 User-defined Functions

udf_memory_limit

Parameter description: Controls the maximum physical memory that can be used when each database node executes UDFs. This parameter does not take effect in the current version. Use **FencedUDFMemoryLimit** and **UDFWorkerMemHardLimit** to control virtual memory used by fenced udf worker.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer. The value range is from 200 x 1024 to *max_process_memory* and the unit is KB.

Default value: **200 MB**

FencedUDFMemoryLimit

Parameter description: Specifies the virtual memory used by each fenced udf worker process.

This parameter is a USERSET parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 KB to 2147483647 KB. The unit can also be MB or GB. **0** indicates that the memory is not limited.

Default value: **0**

UDFWorkerMemHardLimit

Parameter description: Specifies the maximum value of **fencedUDFMemoryLimit**.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0KB to 2147483647KB. The unit can also be MB or GB.

Default value: **1GB**

17.32 Scheduled Task

job_queue_processes

Parameter description: Specifies the number of jobs that can be concurrently executed. This parameter is a POSTMASTER parameter. You can set it using **gs_guc**, and you need to restart **gaussdb** to make the setting take effect.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: 0 to 1000

Function:

- Setting **job_queue_processes** to **0** indicates that the scheduled job function is disabled and that no job will be executed. (Enabling scheduled jobs may affect the system performance. At sites where this function is not required, you are advised to disable it.)
- Setting **job_queue_processes** to a value that is greater than **0** indicates that the scheduled job function is enabled and this value is the maximum number of jobs that can be concurrently processed.

After the scheduled job function is enabled, the job_scheduler thread polls the **pg_job** system catalog at a scheduled interval. The scheduled job check is performed every second by default.

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

Note: If the number of concurrent jobs is large and the value is too small, these jobs will wait in queues. However, a large parameter value leads to large resource consumption. You are advised to set this parameter to **100** and change it based on the system resource condition.

Default value: **10**

enable_prevent_job_task_startup

Parameter description: Specifies whether to start the job thread.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the job thread is not started.
- **off** indicates that the job thread is started.

Default value: **off**

17.33 Thread Pool

The current feature is a lab feature. Contact Huawei technical support before using it.

enable_thread_pool

Parameter description: Specifies whether to enable the thread pool function. This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the thread pool function is enabled.
- **off** indicates that the thread pool function is disabled.

Default value: **on**

thread_pool_attr

Parameter description: Specifies the detailed attributes of the thread pool function. This parameter is valid only when **enable_thread_pool** is set to **on**. Only the sysadmin user can access this parameter. This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string, consisting of one or more characters

This parameter consists of three parts: thread_num, group_num, and cpubind_info. The meanings of the three parts are as follows:

- **thread_num** indicates the total number of threads in the thread pool. The value ranges from 0 to 4096. The value **0** indicates that the database automatically configures the number of threads in the thread pool based on the number of CPU cores. If the value is greater than **0**, the number of threads in the thread pool is the same as the value of **thread_num**. You are advised to set the thread pool size based on the hardware configuration. The formula is as follows: Value of **thread_num** = Number of CPU cores x 3–5. The maximum value of **thread_num** is **4096**.
- **group_num** indicates the number of thread groups in the thread pool. The value ranges from 0 to 64. The value **0** indicates that the database automatically configures the number of thread groups in the thread pool

based on the number of NUMA groups. If the value is greater than **0**, the number of thread groups in the thread pool is the same as the value of **group_num**.

- **cpubind_info** indicates whether the thread pool is bound to a core. The available configuration modes are as follows: 1. **'(nobind)'**: The thread is not bound to a core. 2. **'(allbind)'**: Use all CPU cores that can be queried in the current system to bind threads. 3. **'(nodebind: 1, 2)'**: Use the CPU cores in NUMA groups 1 and 2 to bind threads. 4. **'(cpubind: 0-30)'**: Use CPU cores 0 to 30 to bind threads. 5. **'(numabind: 0-30)'**: Use CPU cores 0 to 30 in the NUMA group to bind threads. This parameter is case-insensitive.

Default value:

'4096,2,(nobind)' (128-core CPU/1024-GB memory, 104-core CPU/1024-GB memory, and 96-core CPU/1024-GB memory); **'2048,2,(nobind)'** (96-core CPU/768-GB memory); **'1024,2,(nobind)'** (64-core CPU/512-GB memory, 60-core CPU/480-GB memory, and 32-core CPU/256-GB memory); **'512,2,(nobind)'** (16-core CPU/128-GB memory); **'256,2,(nobind) '** (8-core CPU/64-GB memory); **'128,2,(nobind)'** (4-core CPU/32-GB memory); **'64,2,(nobind)'** (4-core CPU/16-GB memory)

thread_pool_stream_attr

Parameter description: Specifies the detailed attributes of the stream thread pool function. This parameter is valid only when **enable_thread_pool** is set to **on** and only takes effect on DNs. Only the sysadmin user can access this parameter. This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string, consisting of one or more characters

This parameter consists of four parts: 'stream_thread_num, stream_proc_ratio ,group_num ,cpubind_info'. The meanings of the four parts are as follows:

- **stream_thread_num** indicates the total number of threads in the stream thread pool. The value ranges from 0 to 4096. The value **0** indicates that the database automatically configures the number of threads in the thread pool based on the number of CPU cores. If the value is greater than **0**, the number of threads in the thread pool is the same as the value of **stream_thread_num**. You are advised to set the thread pool size based on the hardware configuration. The formula is as follows: Value of **stream_thread_num** = Number of CPU cores x 3–5. The maximum value of **stream_thread_num** is **4096**.
- **stream_proc_ratio** indicates the ratio of proc resources reserved for stream threads. The value is a floating point number. The default value is **0.2**. The reserved proc resources are calculated as follows: **stream_proc_ratio** x **stream_thread_num**.
- **group_num** indicates the number of thread groups in the thread pool. The value ranges from 0 to 64. The value **0** indicates that the database automatically configures the number of thread groups in the thread pool based on the number of NUMA groups. If the value is greater than **0**, the number of thread groups in the thread pool is the same as the value of **group_num**. The value of **group_num** in **thread_pool_stream_attr** must be

the same as that in **thread_pool_attr**. If they are set to different values, the value of **group_num** in **thread_pool_attr** is used.

- **cpubind_info** indicates whether the thread pool is bound to a core. The available configuration modes are as follows: 1. **'(nobind)'**: The thread is not bound to a core. 2. **'(allbind)'**: Use all CPU cores that can be queried in the current system to bind threads. 3. **'(nodebind: 1, 2)'**: Use the CPU cores in NUMA groups 1 and 2 to bind threads. 4. **'(cpubind: 0-30)'**: Use CPU cores 0 to 30 to bind threads. 5. **'(numabind: 0-30)'**: Use CPU cores 0 to 30 in the NUMA group to bind threads. This parameter is case-insensitive. The value of **cpubind_info** in **thread_pool_stream_attr** must be the same as that in **thread_pool_attr**. If they are set to different values, the value of **cpubind_info** in **thread_pool_attr** is used.

Default value:

stream_thread_num: **16**

stream_proc_ratio: **0.2**

group_num and **cpubind_info**: For details, see **thread_pool_attr**.

resilience_threadpool_reject_cond

Parameter description: Specifies the percentage of accumulated sessions in the thread pool for escape from overload. This parameter takes effect only when the GUC parameters **use_workload_manager** and **enable_thread_pool** are enabled. This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string, consisting of one or more characters

This parameter consists of **recover_threadpool_percent** and **overload_threadpool_percent**. The meanings of the two parts are as follows:

- **recover_threadpool_percent**: Percentage of the number of sessions that are recovered to the normal state in the initial number of threads in the thread pool. When the number of accessed sessions is less than the initial number of threads in the thread pool multiplied by the value of this parameter, the escape from overload function is disabled and new connections are allowed. The value ranges from 0 to **INT_MAX**. The value indicates a percentage.
- **overload_threadpool_percent**: Percentage of the number of accessed sessions to the initial number of threads in the thread pool when the thread pool is overloaded. If the number of accessed sessions is greater than the initial number of threads in the thread pool multiplied by the value of this parameter, the current thread pool is overloaded. In this case, the escape from overload function is enabled to kill sessions and forbid new connections to access the thread pool. The value ranges from 0 to **INT_MAX**. The value indicates a percentage.

Default value: **'0,0'**, indicating that the thread pool escape function is disabled.

Example:

```
resilience_threadpool_reject_cond = '100,200'
```

When the number of stacked sessions exceeds 200% of the initial number of threads in the thread pool, new connections are forbidden and stacked sessions are killed. When the number of stacked sessions is less than 100% of the initial number of threads in the thread pool, new connections are allowed.

NOTICE

- The number of stacked sessions can be obtained by querying the number of data records in the **pg_stat_activity** view. A few background threads need to be filtered out. The initial number of threads in the thread pool can be obtained by querying the **thread_pool_attr** parameter.
- If this parameter is set to a small value, the thread pool escape from overload process is frequently triggered. As a result, ongoing sessions are forcibly logged out, and new connections fail to be connected for a short period of time. Therefore, exercise caution when setting this parameter based on the actual thread pool usage.
- The values of **recover_threadpool_percent** and **overload_threadpool_percent** can be **0** at the same time. In addition, the value of **recover_threadpool_percent** must be smaller than that of **overload_threadpool_percent**. Otherwise, the setting does not take effect.

17.34 Backup and Restoration

operation_mode

Parameter description: Specifies whether the system enters the backup and restoration mode.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the system is in the backup and restoration mode.
- **off** indicates that the system is not in the backup and restoration mode.

Default value: **off**

enable_cbm_tracking

Parameter description: This parameter must be enabled when Roach is used to perform full and incremental backups. If this parameter is disabled, the backup will fail.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on**: The cbm tracking is enabled.
- **off**: The cbm tracking is disabled.

Default value: **off**

17.35 Undo

undo_space_limit_size

Parameter description: Specifies the threshold for forcibly recycling undo space. When the undo space usage reaches 80% of the threshold, forcible recycling starts. You can set this parameter to a large value based on service requirements and then set it to a proper value based on the actual undo space usage.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 800 MB to 16 TB

Default value: **256GB**

undo_limit_size_per_transaction

Parameter description: Specifies the undo space threshold of a single transaction. If the threshold is reached, the transaction is rolled back due to an error.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 2 MB to 16 TB

Default value: **32 GB**

17.36 DCF Parameters Settings

enable_dcf

Parameter description: Specifies whether to enable the DCF mode. This parameter cannot be modified.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean. The value can be **on** or **off**. The value **on** indicates that the current installation mode is DCF, and the value **off** indicates that the current installation mode is not DCF.

Default value: **off**

dcf_ssl

Parameter description: Specifies whether to enable SSL. This parameter takes effect upon the system restart.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean. The value can be **on** or **off**. The value **on** indicates that SSL is used, and the value **off** indicates that SSL is not used.

Default value: **on**

dcf_config

Parameter description: Specifies the customized configuration information during installation. This parameter cannot be modified.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Default value: a string, which is specified by users during installation

dcf_data_path

Parameter description: Specifies the DCF data path. This parameter cannot be modified.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Default value: a string, which is the **dcf_data** directory under the data directory of the DN

dcf_log_path

Parameter description: Specifies the DCF log path. This parameter cannot be modified.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Default value: a string, which is the **dcf_log** directory under the data directory of the DN.

dcf_node_id

Parameter description: Specifies the ID of the DN where the DCF is located. This parameter is defined by users during installation and cannot be modified.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Default value: an integer, which is specified by users during installation

dcf_max_workers

Parameter description: Specifies the number of DCF callback function threads. If the number of nodes exceeds 7, increase the value of this parameter (for example, to **40**). Otherwise, the primary node may remain in the promoting state and the log replication between the primary and standby nodes has no progress.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 10 to 262143

Default value: **10**

dcf_truncate_threshold

Parameter description: Specifies the threshold for a DN to truncate DCF logs.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 1 to 2147483647

Default value: **100000**

dcf_election_timeout

Parameter description: Specifies the timeout interval for selecting the DCF leader and follower. The election timeout interval depends on the status of the network between DNs. If the timeout interval is short and the network quality is poor, timeout occurs. After the network recovers, the election becomes normal. You are advised to set a proper timeout interval based on the current network status. Restriction on the DCF node clock: The maximum clock difference between DCF nodes is less than half of the election timeout period.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 1 to 600, in seconds

Default value: **3**

dcf_enable_auto_election_priority

Parameter description: Specifies whether the DCF priority can be automatically adjusted. The value **0** indicates that automatic adjustment is not allowed, and the value **1** indicates that automatic adjustment is allowed.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: 0 or **1**.

Default value: **1**

dcf_election_switch_threshold

Parameter description: Specifies the DCF threshold for preventing frequent switchover to primary. It is recommended that this parameter be set based on the maximum fault duration acceptable for user services.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 2147483647, in seconds.

Default value: **0**

dcf_run_mode

Parameter description: Specifies the DCF election mode. The value **0** indicates that the automatic DCF election mode is enabled, and the value **2** indicates that the DCF election mode is disabled. Currently, the election mode can be disabled only in minority restoration scenarios. If the election mode is disabled, the database instance will become unavailable.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Enumerated type. The value can be **0** or **2**.

Default value: **0**

dcf_log_level

Parameter description: Specifies the DCF log level.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: a string

- Disable the log function: **NONE**, indicating that the log function is disabled and cannot be used for the following log levels:
- Enable the log function: **RUN_ERR|RUN_WAR|RUN_INF|DEBUG_ERR|DEBUG_WAR|DEBUG_INF|TRACE|PROFILE|OPER**

  You can select a string from the preceding strings and use vertical bars (|) to combine the strings. The log level cannot be left blank.

Default value: **RUN_ERR|RUN_WAR|DEBUG_ERR|OPER|RUN_INF|PROFILE**

dcf_log_backup_file_count

Parameter description: Specifies the number of DCF run log backups.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 1 to 100

Default value: **10**

dcf_max_log_file_size

Parameter description: Specifies the maximum size of a DCF run log file.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 1 to 1000, in MB

Default value: **10**

dcf_socket_timeout

Parameter description: Specifies the timeout interval for the DCF communication module to connect to the socket. This parameter takes effect upon the system restart. In an environment where the network quality is poor, if the timeout interval is set to a small value, a connection may fail to be set up. In this case, you need to increase the value.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 10 to 600000, in ms

Default value: **5000**

dcf_connect_timeout

Parameter description: Specifies the timeout interval for the DCF communication module to set up a connection. This parameter takes effect upon the system restart. In an environment where the network quality is poor, if the timeout interval is set to a small value, the connection may fail to be set up. In this case, you need to increase the value.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 10 to 600000, in ms

Default value: 60000

dcf_mec_fragment_size

Parameter description: Specifies the fragment size of the DCF communication module. This parameter takes effect upon the system restart.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 32 to 10240, in KB

Default value: **64**

dcf_stg_pool_max_size

Parameter description: Specifies the maximum size of the memory pool of the DCF storage module. This parameter takes effect upon the system restart.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 32 to 2147483647, in MB

Default value: **2048**

dcf_stg_pool_init_size

Parameter description: Specifies the minimum size of the memory pool of the DCF storage module. This parameter takes effect upon the system restart.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 32 to 2147483647, in MB

Default value: **32**

dcf_mec_pool_max_size

Parameter description: Specifies the maximum size of the memory pool of the DCF communication module. This parameter takes effect upon the system restart.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 32 to 2147483647, in MB

Default value: 200

dcf_flow_control_disk_rawait_threshold

Parameter description: Specifies the disk waiting threshold for DCF flow control.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 2147483647, in µs

Default value: 100000

dcf_flow_control_net_queue_message_num_threshold

Parameter description: Specifies the threshold for the number of messages in a network queue for DCF flow control.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 2147483647

Default value: 1024

dcf_flow_control_cpu_threshold

Parameter description: Specifies the threshold for DCF CPU flow control.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 2147483647, in percentage (%)

Default value: 100

dcf_mec_batch_size

Parameter description: Specifies the number of batch messages for DCF communication. When the value is **0**, the DCF automatically adjusts the value based on the network and the amount of data to be written. This parameter takes effect upon the system restart.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 1024

Default value: 0

dcf_mem_pool_max_size

Parameter description: Specifies the maximum DCF memory. This parameter takes effect upon the system restart.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 32 to 2147483647, in MB

Default value: **2048**

dcf_mem_pool_init_size

Parameter description: Specifies the initial size of the DCF memory. This parameter takes effect upon the system restart.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 32 to 2147483647, in MB

Default value: **32**

dcf_compress_algorithm

Parameter description: Specifies the compression algorithm for DCF run log transmission. This parameter takes effect upon the system restart.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer

- **0** indicates no compression.
- **1** indicates the ZSTD compression algorithm.
- **2** indicates the LZ4 compression algorithm.

Default value: **0**

dcf_compress_level

Parameter description: Specifies the compression level for DCF log transmission. This parameter takes effect upon the system restart. Before this parameter takes effect, a valid compression algorithm must be configured, that is, the **dcf_compress_algorithm** parameter is set.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 1 to 22

If compression is disabled, the configured compression level does not take effect.

Default value: **1**

dcf_mec_channel_num

Parameter description: Specifies the number of DCF communication channels. This parameter takes effect upon the system restart.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 1 to 64

Default value: **1**

dcf_rep_append_thread_num

Parameter description: Specifies the number of DCF log replication threads. This parameter takes effect upon the system restart.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 1 to 1000

Default value: **2**

dcf_mec_agent_thread_num

Parameter description: Specifies the number of DCF communication working threads. This parameter takes effect upon the system restart. It is recommended that the value of **dcf_mec_agent_thread_num** be greater than or equal to 2 x Number of nodes x Value of **dcf_mec_channel_num**.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 1 to 1000

Default value: **10**

dcf_mec_reactor_thread_num

Parameter description: Specifies the number of reactor threads used by the DCF. This parameter takes effect upon the system restart. It is recommended that the ratio of the value of **dcf_mec_reactor_thread_num** to the value of **dcf_mec_agent_thread_num** be 1:40.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 1 to 100

Default value: **1**

dcf_log_file_permission

Parameter description: Specifies the attribute of the DCF run log file. The parameter setting takes effect after the system is restarted. This parameter is configured during installation and cannot be modified. To allow other users in the same group to access logs, ensure that all parent directories can be accessed by other users in the same group. That is, if **dcf_log_path_permission** is set to **750**, **dcf_log_file_permission** can only be set to **600** or **640**. If **dcf_log_path_permission** is set to **700**, **dcf_log_file_permission** must be set to **600**.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: enumerated type. The value can be **600** or **640**.

Default value: **600**

dcf_log_path_permission

Parameter description: Specifies the attribute of the DCF run log directory. The parameter setting takes effect after the system is restarted. This parameter is configured during installation and cannot be modified. To allow other users in the same group to access the log path, set this parameter to **750**. Otherwise, set this parameter to **700**.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: enumerated type. The value can be **700** or **750**.

Default value: **700**

17.37 Flashback

This section describes parameters related to the flashback function. In this version, only the Ustore engine supports flashback, while the Astore engine does not support flashback.

enable_recyclebin

Parameter description: Specifies whether the recycle bin is enabled or disabled in real time.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

Default value: **off**

recyclebin_retention_time

Parameter description: Specifies the retention period of objects in the recycle bin. The objects will be automatically deleted after the retention period expires.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 1 to 2147483647. The unit is s.

Default value: **15 min** (900s)

version_retention_age

Parameter description: Specifies the number of transactions retained in the old version. If the number of transactions exceeds the value of this parameter, the old version will be recycled and cleared.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 576460752303423487. **0** means no delay.

Default value: **0**

CAUTION

This parameter has been deprecated.

vacuum_defer_cleanup_age

Parameter description: Specifies the number of transactions by which **VACUUM** will defer the cleanup of invalid row-store table records, so that **VACUUM** and **VACUUM FULL** do not clean up deleted tuples immediately. You can also set this parameter to configure the retention period of the flashback function in the old version.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 1000000. **0** means no delay. The value range needs to be extended to 100 million.

Default value: **0**

CAUTION

This parameter can be ignored when you use the Ustore engine to flash back. It serves the Astore flashback function of the earlier version and has other functions. The flashback function is not used in this version.

undo_retention_time

Parameter description: Specifies the period for retaining undo logs of earlier versions.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 0 to 259200. The unit is second.

Default value: **0**

CAUTION

1. If this parameter is set to **0** during the Ustore flashback query, the snapshot information at the flashback point will be cleared. In earlier versions, no flashback query can be performed. When a flashback query is performed, the error message "cannot find the restore point" is displayed.
2. If the time within which the undo logs of earlier versions need to be retained is time1 and the SQL statement execution time for the flashback query is time2, you need to set **undo_retention_time** to a value greater than time1 + time2. That is, set **undo_retention_time** to a value greater than time1 + time2 + 3s. You are advised to set **undo_retention_time** to a value equal to time1 + 1.5 x time2. For example, if you want to retain the logs of earlier versions within the latest 3 hours, and the SQL statement execution time for the flashback query is 1 hour, set **undo_retention_time** to a value equal to 3 hours + 1.5 x 1 hour, that is, 4.5 hours.

17.38 Rollback Parameters

max_undo_workers

Parameter description: Specifies the number of undo worker threads invoked during asynchronous rollback. The parameter setting takes effect after the system is restarted.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

Value range: an integer ranging from 1 to 100

Default value: **5**

17.39 Reserved Parameters

NOTE

The following parameters are reserved and do not take effect in this version.

acce_min_datasize_per_thread

cstore_insert_mode

enable_constraint_optimization

enable_hadoop_env

enable_hdfs_predicate_pushdown

enable_orc_cache

schedule_splits_threshold

backend_version

undo_zone_count

version_retention_age

17.40 Global SysCache Parameters

enable_global_syscache

Parameter description: Specifies whether to enable the global system cache function. This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 10-1**.

Value range: Boolean

- **on** indicates that the global system cache function is enabled.
- **off** indicates that the global system cache function is disabled.

Default value: **on**

You are advised to use this parameter together with the thread pool parameter. After this parameter is enabled, you are advised to set **wal_level** of the standby node to **hot_standby** or higher if you need to access the standby node.

global_syscache_threshold

Parameter description: Specifies the maximum memory usage of the global system cache.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 10-1**.

The **enable_global_syscache** parameter must be enabled.

Value range: an integer ranging from 16384 to 1073741824. The unit is KB.

Default value: **163840**

Recommended calculation formula: The smaller value of the number of hot databases and the number of threads x Memory size allocated to each database.

That is, **global_syscache_threshold** = **min(count(hot dbs)**, **count(threads))** x **memofdb**.

The number of hot databases refers to the number of frequently accessed databases. In thread pool mode, the number of threads is the sum of the number of threads in the thread pool and the number of background threads. In non-thread pool mode, the number of hot databases is used.

memofdb indicates the average memory allocated to each database. The background noise memory of each database is 2 MB. Each time a table or index is added, 11 KB memory is added.

If this parameter is set to a small value, memory is frequently evicted, and a large number of memory fragments cannot be recycled. As a result, memory control fails.

18 Error Log Reference

18.1 Kernel Error Information

ERRMSG: "unsupported syntax: ENCRYPTED WITH in this operation"

SQLSTATE: 42601

CAUSE: "client encryption feature is not supported this operation."

ACTION: "Check client encryption feature whether supported this operation."

ERRMSG: "invalid grant operation"

SQLSTATE: 0LP01

CAUSE: "Grant options cannot be granted to public."

ACTION: "Grant grant options to roles."

ERRMSG: "unrecognized object kind: %d"

SQLSTATE: XX004

CAUSE: "The object type is not supported for GRANT/REVOKE."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported object types."

ERRMSG: "unrecognized GrantStmt.targtype: %d"

SQLSTATE: XX004

CAUSE: "The target type is not supported for GRANT/REVOKE."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported target types."

ERRMSG: "invalid grant operation"

SQLSTATE: 0LP01

CAUSE: "Grant to public operation is forbidden in security mode."

ACTION: "Don't grant to public in security mode."

ERRMSG: "unrecognized object type"

SQLSTATE: XX004

CAUSE: "The object type is not supported for GRANT/REVOKE."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported object types."

ERRMSG: "invalid grant/revoke operation"

SQLSTATE: 0LP01

CAUSE: "Column privileges are only valid for relations in GRANT/REVOKE."

ACTION: "Use the column privileges only for relations."

ERRMSG: "invalid AccessPriv node"

SQLSTATE: 0LP01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "unrecognized GrantStmt.objtype: %d"

SQLSTATE: XX004

CAUSE: "The object type is not supported for GRANT/REVOKE."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported object types."

ERRMSG: "undefined client master key"

SQLSTATE: 42705

CAUSE: "The client master key does not exist."

ACTION: "Check whether the client master key exists."

ERRMSG: "undefined column encryption key"

SQLSTATE: 42705

CAUSE: "The column encryption key does not exist."

ACTION: "Check whether the column encryption key exists."

ERRMSG: "large object %u does not exist"

SQLSTATE: 42704

CAUSE: "The large object does not exist."

ACTION: "Check whether the large object exists."

ERRMSG: "redundant options"

SQLSTATE: 42601

CAUSE: "The syntax 'schemas' is redundant in ALTER DEFAULT PRIVILEGES statement."

ACTION: "Check ALTER DEFAULT PRIVILEGES syntax."

ERRMSG: "redundant options"

SQLSTATE: 42601

CAUSE: "The syntax 'roles' is redundant in ALTER DEFAULT PRIVILEGES statement."

ACTION: "Check ALTER DEFAULT PRIVILEGES syntax."

ERRMSG: "option '%s' not recognized"

SQLSTATE: 42601

CAUSE: "The option in ALTER DEFAULT PRIVILEGES statement is not supported."

ACTION: "Check ALTER DEFAULT PRIVILEGES syntax."

ERRMSG: "unrecognized GrantStmt.objtype: %d"

SQLSTATE: XX004

CAUSE: "The object type is not supported for ALTER DEFAULT PRIVILEGES."

ACTION: "Check ALTER DEFAULT PRIVILEGES syntax to obtain the supported object types."

ERRMSG: "invalid alter default privileges operation"

SQLSTATE: 0LP01

CAUSE: "Default privileges cannot be set for columns."

ACTION: "Check ALTER DEFAULT PRIVILEGES syntax."

ERRMSG: "unrecognized objtype: %d"

SQLSTATE: XX004

CAUSE: "The object type is not supported for default privileges."

ACTION: "Check ALTER DEFAULT PRIVILEGES syntax to obtain the supported object types."

ERRMSG: "could not find tuple for default ACL %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "unexpected default ACL type: %d"

SQLSTATE: 0LP01

CAUSE: "The object type is not supported for default privilege."

ACTION: "Check ALTER DEFAULT PRIVILEGES syntax to obtain the supported object types."

ERRMSG: "invalid object id"

SQLSTATE: 0LP01

CAUSE: "The object type is not supported for GRANT/REVOKE."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported object types."

ERRMSG: "undefined column"

SQLSTATE: 42703

CAUSE: "The column of the relation does not exist."

ACTION: "Check whether the column exists."

ERRMSG: "column number out of range"

SQLSTATE: 0LP01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "cache lookup failed for attribute %d of relation %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "cache lookup failed for relation %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "unsupported object type"

SQLSTATE: 42809

CAUSE: "Index type is not supported for GRANT/REVOKE."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported object types."

ERRMSG: "unsupported object type"

SQLSTATE: 42809

CAUSE: "Composite type is not supported for GRANT/REVOKE."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported object types."

ERRMSG: "wrong object type"

SQLSTATE: 42809

CAUSE: "GRANT/REVOKE SEQUENCE only support sequence objects."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported object types."

ERRMSG: "invalid privilege type USAGE for table"

SQLSTATE: 0LP01

CAUSE: "GRANT/REVOKE TABLE do not support USAGE privilege."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported privilege types for tables."

ERRMSG: "invalid privilege type %s for column"

SQLSTATE: 0LP01

CAUSE: "The privilege type is not supported for column object."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported privilege types for column object."

ERRMSG: "cache lookup failed for database %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "cache lookup failed for foreign-data wrapper %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "cache lookup failed for foreign server %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "cache lookup failed for function %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "cache lookup failed for language %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "Grant/revoke on untrusted languages if forbidden."

SQLSTATE: 0LP01

CAUSE: "Grant/revoke on untrusted languages if forbidden."

ACTION: "Support grant/revoke on trusted C languages"

ERRMSG: "Forbid grant language c to user with grant option."

SQLSTATE: 0A000

CAUSE: "Forbid grant language c to user with grant option."

ACTION: "Only support grant language c to user."

ERRMSG: "Forbid grant language c to public."

SQLSTATE: 0A000

CAUSE: "Forbid grant language c to public."

ACTION: "Grant language c to specified users."

ERRMSG: "cache lookup failed for large object %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "cache lookup failed for namespace %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "cache lookup failed for tablespace %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "cache lookup failed for type %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "cannot set privileges of array types"

SQLSTATE: 0LP01

CAUSE: "Cannot set privileges of array types."

ACTION: "Set the privileges of the element type instead."

ERRMSG: "wrong object type"

SQLSTATE: 42809

CAUSE: "GRANT/REVOKE DOMAIN only support domain objects."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported object types."

ERRMSG: "cache lookup failed for data source %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "cache lookup failed for client master key %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "cache lookup failed for column encryption key %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "cache lookup failed for directory %u"

SQLSTATE: 29P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "unrecognized privilege type '%s'"

SQLSTATE: 42601

CAUSE: "The privilege type is not supported."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported privilege types."

ERRMSG: "unrecognized privilege: %d"

SQLSTATE: XX004

CAUSE: "The privilege type is not supported."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported privilege types."

ERRMSG: "unrecognized AclResult"

SQLSTATE: XX004

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "permission denied for column '%s' of relation '%s'"

SQLSTATE: 42501

CAUSE: "Insufficient privileges for the column."

ACTION: "Select the system tables to get the acl of the column."

ERRMSG: "role with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "unrecognized objkind: %d"

SQLSTATE: XX004

CAUSE: "The object type is not supported for privilege check."

ACTION: "Check GRANT/REVOKE syntax to obtain the supported object types."

ERRMSG: "attribute %d of relation with OID %u does not exist"

SQLSTATE: 42703

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "the column has been dropped"

SQLSTATE: 42703

CAUSE: "The column does not exist."

ACTION: "Check whether the column exists."

ERRMSG: "relation with OID %u does not exist"

SQLSTATE: 42P01

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "invalid group"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "database with OID %u does not exist"

SQLSTATE: 3D000

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "directory with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "function with OID %u does not exist"

SQLSTATE: 42883

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "client master key with OID %u does not exist"

SQLSTATE: 42705

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "language with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "large object %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "schema with OID %u does not exist"

SQLSTATE: 3F001

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "tablespace with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "foreign-data wrapper with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "foreign server with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "data source with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "type with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "operator with OID %u does not exist"

SQLSTATE: 42883

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "column encryption key with OID %u does not exist"

SQLSTATE: 42705

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "operator class with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "operator family with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "text search dictionary with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "text search configuration with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "collation with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "conversion with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "extension with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "synonym with OID %u does not exist"

SQLSTATE: 42704

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "package can not create the same name with schema."

SQLSTATE: 22023

CAUSE: "Package name conflict"

ACTION: "Please rename package name"

ERRMSG: "type is not exists %s."

SQLSTATE: 22023

CAUSE: "System error."

ACTION: "Contact Huawei Engineer."

ERRMSG: "This input type is not supported for tdigest_in()"

SQLSTATE: 0A000

CAUSE: "input type is not supported"

ACTION: "Check tdigest_in syntax to obtain the supported privilege types"

ERRMSG: "Failed to apply for memory"

SQLSTATE: 53200

CAUSE: "palloc failed"

ACTION: "Check memory"

ERRMSG: "Failed to get tde info from relation '%s'."

SQLSTATE: XX005

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "SPI_connect failed: %s"

SQLSTATE: SP001

CAUSE: "System error."

ACTION: "Analyze the error message before the error"

ERRMSG: "permission denied for terminate snapshot thread"

SQLSTATE: 42501

CAUSE: "The user does not have system admin privilege"

ACTION: "Grant system admin to user"

ERRMSG: "terminate snapshot thread failed"

SQLSTATE: OP001

CAUSE: "Execution failed due to: %s"

ACTION: "check if snapshot thread exists"

ERRMSG: "terminate snapshot thread failed"

SQLSTATE: OP001

CAUSE: "restart wdr snapshot thread timeoutor The thread did not respond to the kill signal"

ACTION: "Check the wdr snapshot thread is restarted"

ERRMSG: "set lockwait_timeout failed"

SQLSTATE: XX000

CAUSE: "System error."

ACTION: "Contact engineer to support."

ERRMSG: "permission denied for create WDR Snapshot"

SQLSTATE: 42501

CAUSE: "The user does not have system admin privilege"

ACTION: "Grant system admin to user"

ERRMSG: "WDR snapshot request can not be accepted, please retry later"

SQLSTATE: OP001

CAUSE: "wdr snapshot thread does not exist"

ACTION: "Check if wdr snapshot thread exists"

ERRMSG: "Cannot respond to WDR snapshot request"

SQLSTATE: OP001

CAUSE: "Execution failed due to: %s"

ACTION: "Check if wdr snapshot thread exists"

ERRMSG: "query(%s) can not get datum values"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the query can be executed"

ERRMSG: "create sequence failed"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check if sequence can be created"

ERRMSG: "update snapshot end time stamp filled"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the snapshot retry is successful"

ERRMSG: "query can not get datum values"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the query can be executed"

ERRMSG: "SPI_connect failed: %s"

SQLSTATE: XX000

CAUSE: "System error."

ACTION: "Check whether the snapshot retry is successful"

ERRMSG: "query(%s) execute failed"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the snapshot retry is successful"

ERRMSG: "clean table of snap_%s is failed"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the snapshot retry is successful"

ERRMSG: "analyze table failed"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the snapshot retry is successful"

ERRMSG: "insert into tables_snap_timestamp start time stamp is failed"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the snapshot retry is successful"

ERRMSG: "insert data failed"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the snapshot retry is successful and check whether the query can be executed"

ERRMSG: "update tables_snap_timestamp end time stamp is failed"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the snapshot retry is successful"

ERRMSG: "clean snapshot id %lu is failed in snapshot table"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the snapshot retry is successful and check whether the query can be executed"

ERRMSG: "clean snapshot failed"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the snapshot retry is successful"

ERRMSG: "can not create snapshot stat table"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the query can be executed"

ERRMSG: "create WDR snapshot data table failed"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the query can be executed"

ERRMSG: "insert into tables_snap_timestamp start time stamp failed"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the query can be executed"

ERRMSG: "insert into snap_%s is failed"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the query can be executed"

ERRMSG: "update tables_snap_timestamp end time stamp failed"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the query can be executed"

ERRMSG: "create index failed"

SQLSTATE: 22000

CAUSE: "System error."

ACTION: "Check whether the query can be executed"

ERRMSG: "analyze table, connection failed: %s"

SQLSTATE: XX000

CAUSE: "System error."

ACTION: "Check whether the snapshot retry is successful"

ERRMSG: "snapshot thread SPI_connect failed: %s"

SQLSTATE: XX000

CAUSE: "System error."

ACTION: "Check whether the snapshot retry is successful"

ERRMSG: "Distributed key column can't be transformed"

SQLSTATE: 42P10

CAUSE: "There is a risk of violating uniqueness when transforming distribution columns."

ACTION: "Change transform column."

ERRMSG: "cannot convert %s to %s"

SQLSTATE: 42804

CAUSE: "There is no conversion path in pg_cast."

ACTION: "Rewrite or cast the expression."

ERRMSG: "create matview on TDE table failed"

SQLSTATE: 0A000

CAUSE: "create materialized views is not supported on TDE table"

ACTION: "check CREATE syntax about create the materialized views"

ERRMSG: "schema name can not same as package"

SQLSTATE: 22023

CAUSE: "schema name conflict"

ACTION: "rename schema name"

ERRMSG: "Unrecognized commandType when checking read-only attribute."

SQLSTATE: XX004

CAUSE: "System error."

ACTION: "Contact Huawei Engineer."

ERRMSG: "Fail to generate subquery plan."

SQLSTATE: XX005

CAUSE: "System error."

ACTION: "Contact Huawei Engineer."

ERRMSG: "Unrecognized node type when processing qual condition."

SQLSTATE: XX004

CAUSE: "System error."

ACTION: "Contact Huawei Engineer."

ERRMSG: "Unrecognized node type when processing const parameters."

SQLSTATE: XX004

CAUSE: "System error."

ACTION: "Contact Huawei Engineer."

ERRMSG: "SELECT FOR UPDATE/SHARE is not allowed with UNION/INTERSECT/EXCEPT"

SQLSTATE: 0A000

CAUSE: "SQL uses unsupported feature."

ACTION: "Modify SQL statement according to the manual."

ERRMSG: "GROUP BY cannot be implemented."

SQLSTATE: 0A000

CAUSE: "GROUP BY uses unsupported datatypes."

ACTION: "Modify SQL statement according to the manual."

ERRMSG: "TSDB functions cannot be used if enable_tsdb is off."

SQLSTATE: D0011

CAUSE: "Functions are not loaded."

ACTION: "Turn on enable_tsdb according to manual."

ERRMSG: "Unrecognized node type when extracting index."

SQLSTATE: XX004

CAUSE: "System error."

ACTION: "Contact Huawei Engineer."

ERRMSG: "Ordering operator cannot be identified."

SQLSTATE: 42883

CAUSE: "Grouping set columns must be able to sort their inputs."

ACTION: "Modify SQL statement according to the manual."

ERRMSG: "DISTINCT cannot be implemented."

SQLSTATE: 0A000

CAUSE: "DISTINCT uses unsupported datatypes."

ACTION: "Modify SQL statement according to the manual."

ERRMSG: "Failed to locate grouping columns."

SQLSTATE: 55000

CAUSE: "System error."

ACTION: "Contact Huawei Engineer."

ERRMSG: "Resjunk output columns are not implemented."

SQLSTATE: 20000

CAUSE: "System error."

ACTION: "Contact Huawei Engineer."

ERRMSG: "PARTITION BY cannot be implemented."

SQLSTATE: 0A000

CAUSE: "PARTITION BY uses unsupported datatypes."

ACTION: "Modify SQL statement according to the manual."

ERRMSG: "ORDER BY cannot be implemented."

SQLSTATE: 0A000

CAUSE: "ORDER BY uses unsupported datatypes."

ACTION: "Modify SQL statement according to the manual."

ERRMSG: "Failed to deconstruct sort operators into partitioning/ordering operators."

SQLSTATE: D0011

CAUSE: "System error."

ACTION: "Contact Huawei Engineer."

ERRMSG: "OBS and HDFS foreign table can NOT be in the same plan."

SQLSTATE: XX008

CAUSE: "SQL uses unsupported feature."

ACTION: "Modify SQL statement according to the manual."

ERRMSG: "Pool size should not be zero"

SQLSTATE: 22012

CAUSE: "Compute pool configuration file contains error."

ACTION: "Please check the value of 'pl' in cp_client.conf."

ERRMSG: "Failed to get the runtime info from the compute pool."

SQLSTATE: 22004

CAUSE: "System error."

ACTION: "Contact Huawei Engineer."

ERRMSG: "Version is not compatible between local cluster and the compute pool."

SQLSTATE: XX008

CAUSE: "Compute pool is not installed appropriately."

ACTION: "Configure compute pool according to manual."

ERRMSG: "No optional index path is found."

SQLSTATE: 01000

CAUSE: "System error."

ACTION: "Contact Huawei Engineer."

ERRMSG: "MERGE INTO on replicated table does not yet support using distributed tables."

SQLSTATE: 0A000

CAUSE: "SQL uses unsupported feature."

ACTION: "Modify SQL statement according to the manual."

ERRMSG: "Fail to find ForeignScan node!"

SQLSTATE: P0002

CAUSE: "System error."

ACTION: "Contact Huawei Engineer."

ERRMSG: "sql advisor don't support none table, temp table, system table."

SQLSTATE: 42601

CAUSE: "sql advisor don't support none table, temp table, system table."

ACTION: "check query component"

ERRMSG: "Invalid autonomous transaction return datatypes"

SQLSTATE: P0000

CAUSE: "PL/SQL uses unsupported feature."

ACTION: "Contact Huawei Engineer."

ERRMSG: "new row for relation '%s' violates check constraint '%s'"

SQLSTATE: 23514

CAUSE: "some rows copy failed"

ACTION: "check table definition"

ERRMSG: "new row for relation '%s' violates check constraint '%s'"

SQLSTATE: 23514

CAUSE: "some rows copy failed"

ACTION: "set client_min_messages = info for more details"

ERRMSG: "get gauss home path is NULL"

SQLSTATE: XX005

CAUSE: "gauss home path not set"

ACTION: "check if $GAUSSHOME is exist"

ERRMSG: "unable to open kms_iam_info.json file"

SQLSTATE: 58P03

CAUSE: "file not exist or broken"

ACTION: "check the kms_iam_info.json file"

ERRMSG: "can not get password plaintext"

SQLSTATE: XX005

CAUSE: "file not exist or broken"

ACTION: "check the password cipher rand file"

ERRMSG: "IAM info json key is NULL"

SQLSTATE: XX005

CAUSE: "IAM info value error"

ACTION: "check tde_config kms_iam_info.json file"

ERRMSG: "get internal password is NULL"

SQLSTATE: XX005

CAUSE: "cipher rand file missing"

ACTION: "check password cipher rand file"

ERRMSG: "KMS info json key is NULL"

SQLSTATE: XX005

CAUSE: "KMS info value error"

ACTION: "check tde_config kms_iam_info.json file"

ERRMSG: "unable to get json file"

SQLSTATE: 58P03

CAUSE: "parse json file failed"

ACTION: "check the kms_iam_info.json file format"

ERRMSG: "get JSON tree is NULL"

SQLSTATE: XX005

CAUSE: "get KMS JSON tree failed"

ACTION: "check input parameter or config.ini file"

ERRMSG: "failed to get json tree"

SQLSTATE: XX005

CAUSE: "config.ini json tree error"

ACTION: "check input parameter or config.ini file"

ERRMSG: "failed to set the value of json tree"

SQLSTATE: XX005

CAUSE: "config.ini json tree error"

ACTION: "check input parameter or config.ini file"

ERRMSG: "http request failed"

SQLSTATE: XX005

CAUSE: "http request error"

ACTION: "check KMS or IAM connect or config parameter"

ERRMSG: "get iam token or iam agency token is NULL"

SQLSTATE: XX005

CAUSE: "connect IAM failed"

ACTION: "check if your env can connect with IAM server"

ERRMSG: "KMS dek json key is NULL"

SQLSTATE: XX005

CAUSE: "KMS return value error"

ACTION: "check KMS config parameter"

ERRMSG: "get kms dek is NULL"

SQLSTATE: XX005

CAUSE: "connect KMS failed"

ACTION: "check if your env can connect with KMS server"

ERRMSG: "get http header is NULL"

SQLSTATE: XX005

CAUSE: "http request failed"

ACTION: "check IAM config parameter"

ERRMSG: "create KMS dek failed"

SQLSTATE: XX005

CAUSE: "KMS error"

ACTION: "check KMS connect or config parameter"

ERRMSG: "get KMS dek failed"

SQLSTATE: XX005

CAUSE: "KMS error"

ACTION: "check KMS connect or config parameter"

ERRMSG: "get KMS DEK is NULL"

SQLSTATE: XX005

CAUSE: "get KMS dek_plaintext failed"

ACTION: "check KMS network or cipher is right"

ERRMSG: "create matview with TDE failed"

SQLSTATE: 0A000

CAUSE: "TDE feature is not supported for Create materialized views"

ACTION: "check CREATE syntax about create the materialized views"

ERRMSG: "failed to add item to the index page"

SQLSTATE: XX002

CAUSE: "System error."

ACTION: "Check WARNINGS for the details."

ERRMSG: "index row size %lu exceeds maximum %lu for index '%s'"

SQLSTATE: 54000

CAUSE: "Values larger than 1/3 of a buffer page cannot be indexed."

ACTION: "Consider a function index of an MD5 hash of the value, or use full text indexing."

18.2 CM Error Information

ERRMSG: "Fail to access the cluster static config file."

SQLSTATE: c3000

CAUSE: "The cluster static config file is not generated or is manually deleted."

ACTION: "Please check the cluster static config file."

ERRMSG: "Fail to open the cluster static file."

SQLSTATE: c3000

CAUSE: "The cluster static config file is not generated or is manually deleted."

ACTION: "Please check the cluster static config file."

ERRMSG: "Fail to read the cluster static file."

SQLSTATE: c3001

CAUSE: "The cluster static file permission is insufficient."

ACTION: "Please check the cluster static config file."

ERRMSG: "Failed to read the static config file."

SQLSTATE: c1000

CAUSE: "out of memeory."

ACTION: "Please check the system memory and try again."

ERRMSG: "Could not find the current node in the cluster by the node id %u."

SQLSTATE: c3002

CAUSE: "The static config file probably contained content error."

ACTION: "Please check static config file."

ERRMSG: "Failed to open the logic config file."

SQLSTATE: c3000

CAUSE: "The logic config file is not generated or is manually deleted."

ACTION: "Please check the cluster static config file."

ERRMSG: "Fail to read the logic static config file."

SQLSTATE: c3001

CAUSE: "The logic static config file permission is insufficient."

ACTION: "Please check the logic static config file."

ERRMSG: "Failed to open or read the static config file."

SQLSTATE: c1000

CAUSE: "out of memeory."

ACTION: "Please check the system memory and try again."

ERRMSG: "Failed to open the log file '%s'."

SQLSTATE: c3000

CAUSE: "Log file not found."

ACTION: "Please check the log file."

ERRMSG: "Failed to open the log file '%s'."

SQLSTATE: c3000

CAUSE: "The log file permission is insufficient."

ACTION: "please check the log file."

ERRMSG: "Failed to open the dynamic config file '%s'."

SQLSTATE: c3000

CAUSE: "The dynamic config file permission is insufficient."

ACTION: "Please check the dynamic config file."

ERRMSG: "Failed to malloc memory, size = %lu."

SQLSTATE: c1000

CAUSE: "out of memeory."

ACTION: "Please check the system memory and try again."

ERRMSG: "unrecognized AZ name '%s'."

SQLSTATE: c3000

CAUSE: "The parameter(%s) entered by the user is incorrect."

ACTION: "Please check the parameter entered by the user and try again."

ERRMSG: "unrecognized minorityAz name '%s'."

SQLSTATE: c3000

CAUSE: "The parameter(%s) entered by the user is incorrect."

ACTION: "Please check the parameter entered by the user and try again."

ERRMSG: "Get GAUSSHOME failed."

SQLSTATE: c3000

CAUSE: "The environment variable('GAUSSHOME') is incorrectly configured."

ACTION: "Please check the environment variable('GAUSSHOME')."

ERRMSG: "Get current user name failed."

SQLSTATE: c3000

CAUSE: "N/A"

ACTION: "Please check the environment."

ERRMSG: "-B option must be specified."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-T option must be specified.\n"

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "can't stop one node or instance with -m normal."

SQLSTATE: c3000

CAUSE: "The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "can't stop one node or instance with -m resume."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "can't stop one availability zone with -m resume."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "log level or cm server arbitration mode must be specified."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "log level or cm server arbitration mode need not be specified."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-R is needed."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-D is needed."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-n and -R are needed."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-n and -D are needed."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "no operation specified."

SQLSTATE: c3000

CAUSE: "The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "no cm directory specified."

SQLSTATE: c3000

CAUSE: "The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "Please check the usage of switchover."

SQLSTATE: c3000

CAUSE: "The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-n and -z cannot be specified at the same time."

SQLSTATE: c3000

CAUSE: "The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-m cannot be specified at the same time with -n or -z."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-n node(%d) is invalid."

SQLSTATE: c3000

CAUSE: "The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-n node is needed."

SQLSTATE: c3000

CAUSE: "The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "%s: -C is needed."

SQLSTATE: c3000

CAUSE: "The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-z value must be 'ALL' when query mppdb cluster."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-v is needed."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-C is needed."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-Cv is needed."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-L value must be 'ALL' when query logic cluster."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "unrecognized LC name '%s'."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-n is needed."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "There is no '%s' information in cluster."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-D path is too long.\n"

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-D path is invalid."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-n node(%s) is invalid."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-R only support when the cluster is single-inst."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-t time is invalid."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "-votenum is invalid."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "unrecognized build mode."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "unrecognized build mode '%s'."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "too many command-line arguments (first is '%s')."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "unrecognized operation mode '%s'."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "no cm directory specified."

SQLSTATE: c3000

CAUSE: "%s: The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "Failed to malloc memory."

SQLSTATE: c1000

CAUSE: "out of memeory."

ACTION: "Please check the system memory and try again."

ERRMSG: "Failed to open etcd: %s."

SQLSTATE: c4000

CAUSE: "Etcd is abnoraml."

ACTION: "Please check the Cluster Status and try again."

ERRMSG: "[PATCH-ERROR] hotpatch command or path set error."

SQLSTATE: c3000

CAUSE: "The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "no standby datanode in single node cluster."

SQLSTATE: c3000

CAUSE: "The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "restart logic cluster failed."

SQLSTATE: c3000

CAUSE: "The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "restart logic cluster failed"

SQLSTATE: c3000

CAUSE: "The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."

ERRMSG: "The option parameter is not specified."

SQLSTATE: c3000

CAUSE: "The cmdline entered by the user is incorrect."

ACTION: "Please check the cmdline entered by the user(%s)."