Data Warehouse Service

Developer Guide

Issue 03

Date 2024-12-18

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Contents

1 Before You Start

Target Readers

This document is intended for database designers, application developers, and database administrators, and provides information required for designing, building, querying and maintaining data warehouses.

As a database administrator or application developer, you need to be familiar with:

- Knowledge about OSs, which is the basis for everything.
- SQL syntax, which is the necessary skill for database operation.

Prerequisites

Complete the following tasks before you perform operations described in this document:

- Create a GaussDB(DWS) cluster.
- Install a SQL client.
- Connect the SQL client to the default database of the cluster.

For details about these tasks, see **Getting Started with GaussDB(DWS)**.

Reading Guide

If you are a new GaussDB(DWS) user, you are advised to read the following contents first:

- Sections describing the features, functions, and application scenarios of GaussDB(DWS).
- "Getting Started": guides you through creating a data warehouse cluster, creating a database table, uploading data, and testing queries.

If you intend to or are migrating applications from other data warehouses to GaussDB(DWS), you might want to know how GaussDB(DWS) differs from them.

You can find useful information from the following table for GaussDB(DWS) database application development.

Operation	Query Suggestion
Quickly getting started with GaussDB(DWS)	Deploy a cluster, connect to the database, and perform some queries by referring to . First, follow the steps in **Getting Started with GaussDB(DWS)** to quickly deploy a cluster, connect to a database, and try some queries. When you are ready to construct a database, load data to tables and compile the query content to operate the data in the data warehouse. Then, you can return to the *Data Warehouse Service Database Developer Guide*.
Understand the internal architecture of a GaussDB(DWS) data warehouse.	To know more about GaussDB(DWS), go to the GaussDB(DWS) homepage.
Learn how to design tables to achieve the excellent performance.	**GaussDB(DWS) Development Design Specifications** introduces the design specifications that should be complied with during the development of database applications. Modeling compliant with these specifications fits the distributed processing architecture of GaussDB(DWS) and provides efficient SQL code. To facilitate service execution through optimization, you can refer to **Overview of Query Performance Optimization**. Database administrators' experience and judgment play a more significant role in achieving successful performance optimization than instructions and explanations. However, **Overview of Query Performance Optimization** still tries to systematically illustrate the performance optimization methods for application development personnel and new GaussDB(DWS) database administrators.
Loading data	**Importing Data** describes how to import data to GaussDB(DWS). **Importing Best Practices** provides experience tips for fast and efficient data import.
Managing users, groups, and database security	**GaussDB(DWS) Database Security Management** covers database security topics.
Monitoring and optimizing system performance	**GaussDB(DWS) System Catalogs and Views** describes the system catalogs where you can query the database status and monitor the query content and process. You should also refer to **Management Guide** to learn how to use the GaussDB (DWS) console to check the system running status and monitoring metrics.

SQL Syntax Text Conventions

To better understand how to use the syntax, you can refer to the following description of SQL syntax text conventions.

Format	Description
Uppercase characters	Keywords must be in uppercase.
Lowercase characters	Parameters must be in lowercase.
[ ]	Items in brackets [] are optional.
...	Preceding elements can appear repeatedly.
[x | y | ...]	One item is selected from two or more options or no item is selected.
{ x | y | ... }	One item is selected from two or more options.
[x | y | ...] [...]	You can choose either multiple parameters or no parameters. If you choose multiple parameters, simply separate them with spaces.
[x | y | ...] [,...]	You can choose either multiple parameters or no parameters. If you choose multiple parameters, simply separate them with commas (,).
{ x | y | ... } [...]	You must select at least one parameter. If you select multiple parameters, separate them with spaces.
{ x | y | ... } [,...]	You must select at least one parameter. If you select multiple parameters, separate them with commas (,).

Statement

When writing documents, the writers of GaussDB(DWS) try their best to provide guidance from the perspective of commercial use, application scenarios, and task completion. Even so, references to PostgreSQL content may still exist in the document. For this type of content, the following PostgreSQL Copyright is applicable:

Postgres-XC is Copyright © 1996-2013 by the PostgreSQL Global Development Group.

PostgreSQL is Copyright © 1996-2013 by the PostgreSQL Global Development Group.

Postgres95 is Copyright © 1994-5 by the Regents of the University of California.

IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING LOST PROFITS, ARISING OUT OF THE USE OF THIS SOFTWARE AND

2 GaussDB(DWS) Development Design Specifications

2.1 Overview

Objective

This document outlines the rules for design and development that need to be followed when developing the GaussDB(DWS) database. The objective is to enhance development efficiency and ensure the continuity and stability of the service.

Application Scope

These specifications apply to all GaussDB(DWS) self-development scenarios, including designing and developing applications and database services.

Terms

Rule: a mandatory requirement that must be followed during database design and development.

Suggestion: an option that you need to consider for the design and development process.

Description: a detailed explanation of a rule or suggestion.

Overall Development and Design Specifications

The table below provides a list of development and design specifications that must be followed during GaussDB(DWS) development. You can click the links to access the corresponding rules for more details.

Table 2-1 GaussDB(DWS) development and design specifications

No.	Category		Rule/Suggestion
1	Connection management regulations	-	**Rule 1.1: Configuring Load Balancing for GaussDB(DWS) Clusters**
2			**Rule 1.2: Ending the Database Connection After Necessary Operations (Except in Connection Pool Scenarios)**
3			**Rule 1.3: Ensuring a Started Transaction Is Committed or Rolled Back**
4			**Rule 1.4: Ensuring the Idle Timeout Duration Is Shorter Than SESSION_TIMEOUT Value When Connection Pool Is Used for Applications**
5			**Rule 1.5: Restoring Parameters to Default Values in Connections Before Returning Them to the Pool**
6			**Rule 1.6: Manually Clearing Temporary Tables Created with a Connection Before Returning it to the Pool**
7	Object design specifications	DATABASE object design	**Rule 2.1: Avoiding Direct Usage of Built-in Databases Such As postgres and gaussdb**
8			**Rule 2.2: Selecting the Suitable Database Code During Database Creation**
9			**Rule 2.3: Choosing the Right Database Type for Compatibility with the Database to Be Created**
10			**Suggestion 2.4: Storing Objects with Associated Calculations in the Same Database**
11		USER object design	**Rule 2.5: Following the Least Privilege Principle and Avoiding Running Services Using Users with Special Permissions**
12			**Rule 2.6: Avoiding the Use of a Single Database Account for All Services**
13		SCHEMA object design	**Suggestion 2.7: Avoiding the Creation of Objects Under Other Users' Private Schemas**
14		TABLESPACE object design	**Rule 2.8 Avoiding Tablespace Customization**

No.	Category		Rule/Suggestion
15		TABLE object design (prioritized)	**Rule 2.9: Selecting the Optimal Distribution Method and Columns During Table Creation**
16			**Rule 2.10 Selecting an Optimal Storage Type During Table Creation**
17			**Rule 2.11 Selecting an Optimal Partitioning Policy During Table Creation**
18			**Suggestion 2.12: Designing Table Columns for Fast and Accurate Queries**
19			**Suggestion 2.13: Avoiding the Usage of Auto-increment Columns or Data Types**
20		INDEX object design (prioritized)	**Rule 2.14: Creating Necessary Indexes and Selecting Optimal Columns and Sequences for Them**
21			**Suggestion 2.15: Optimizing Performance by Choosing the Right Index Type and Avoiding Indexes for Column-Store Tables**
22		VIEW object design	**Suggestion 2.16: Limiting View Nesting to Three Layers**
23	SQL development specifications	DDL operation specifications	**Suggestion 3.1: Avoiding Performing DDL Operations (Except CREATE) During Peak Hours or in Long Transactions**
24			**Rule 3.2: Specifying the Scope of Objects to Be Deleted When Using DROP**
25		INSERT operation specifications	**Rule 3.3: Replacing INSERT with COPY for Efficient Multi-Value Batch Insertion**
26			**Suggestion 3.4: Avoiding Performing Real-time INSERT Operations on Common Column-store Tables**
27		UPDATE/DELETE operation specifications	**Suggestion 3.5: Preventing Simultaneous Updates or Deletions of the Same Row in a Row-store Table**
28			**Suggestion 3.6: Avoiding Frequent or Simultaneous UPDATE and DELETE Operations on Column-store Tables**
29		SELECT operation specifications	**Rule 3.7: Avoiding Executing SQL Statements That Do Not Support Pushdown**
30			**Rule 3.8: Specifying Association Conditions when Multiple Tables Are Associated**

No.	Category		Rule/Suggestion
31			Rule 3.9: Ensuring Consistency of Data Types in Associated Fields across Multiple Tables
32			Suggestion 3.10: Avoiding Function Calculation on Association and Filter Condition Fields
33			Suggestion 3.11: Performing Pressure Tests and Concurrency Control for Resource-intensive SQL Statements
34			Rule 3.12: Avoiding Excessive COUNT Operations on Large Row-store Tables
35			Suggestion 3.13: Avoid Getting Large Result Sets (Except for Data Exports)
36			Suggestion 3.14: Avoiding the Usage of SELECT * for Queries
37			Suggestion 3.15: Using WITH RECURSIVE with Defined Termination Condition for Recursion
38			Suggestion 3.16: Setting Schema Prefix for Table and Function Access
39			Suggestion 3.17: Identifying an SQL Statement with a Unique SQL Comment
40	Stored procedure development specifications	-	Suggestion 4.1: Simplifying Stored Procedures and Avoiding Nesting
41			Rule 4.2: Avoiding Non-CREATE DDL Operations in Stored Procedures

2.2 GaussDB(DWS) Connection Management Specifications

Rule 1.1: Configuring Load Balancing for GaussDB(DWS) Clusters

NOTE

Impact of rule violation:

- Load imbalance causes performance problems and even service interruption.
- When a CN is faulty, services cannot be automatically recovered or the recovery may take a long time.

Solution:

- Configure ELB load balancing and connect the application to the load balancing IP address.
- For how to use the JDBC for load balancing, see **Configuring JDBC to Connect to a Cluster (Load Balancing Mode)**.

Rule 1.2: Ending the Database Connection After Necessary Operations (Except in Connection Pool Scenarios)

NOTE

Impact of rule violation:

- The number of idle connections exceeds the maximum limit, causing connection creation failure.
- Resource overload occurs because there are too many idle connections.

Solution:

- After the connection between the application and the database is established and used, manually end the connection.
- Set the **session_timeout** parameter on the service side to set the idle timeout duration. The connection will be automatically ended when the idle timeout duration expires.

Rule 1.3: Ensuring a Started Transaction Is Committed or Rolled Back

NOTE

Impact of rule violation:

- If a transaction remains uncommitted for an extended period, it blocks operations such as **ALTER**, thereby affecting all services.
- The number of idle connections exceeds the maximum limit, causing connection creation failure.

Solution:

- **autocommit** is enabled by default, so there is no need to manually commit any transaction unless you modify the default setting.
- If a transaction is explicitly started, it must be explicitly ended (either by committing or rolling back) once the relevant operations are finished.

Rule 1.4: Ensuring the Idle Timeout Duration Is Shorter Than SESSION_TIMEOUT Value When Connection Pool Is Used for Applications

NOTE

Impact of rule violation:

- The idle timeout mechanism on the service side clears connections in the connection pool, which negatively impacts connection reuse.

Solution:

- To ensure everything works correctly, make sure the idle timeout duration of the connection pool is shorter than the **SESSION_TIMEOUT** value in GaussDB(DWS). It is advised to adjust the idle timeout duration rather than modifying the **SESSION_TIMEOUT** value.

Rule 1.5: Restoring Parameters to Default Values in Connections Before Returning Them to the Pool

NOTE

Impact of rule violation:

- When a connection is reused by another service, the parameters set by the service may also be reused. This can result in performance issues or service errors.

Solution:

- Before returning the connection to the connection pool, use **SET SESSION AUTHORIZATION DEFAULT;RESET ALL;** to reset parameters.

Notes:

When connection pool is used for applications, if you set the global GUC parameter using **GS_GUC RELOAD** in GaussDB(DWS), restart the connection pool for the changes to be applied. This is because the modification only affects new connections in the connection pool.

Rule 1.6: Manually Clearing Temporary Tables Created with a Connection Before Returning it to the Pool

NOTE

Impact of rule violation:

- When a connection is reused by other services, an error may be reported when a temporary table is created.

Solution:

- Before returning a connection to the connection pool, use **DROP** to clear the temporary table created by the current session.

2.3 GaussDB(DWS) Object Design Specifications

2.3.1 DATABASE Object Design

Rule 2.1: Avoiding Direct Usage of Built-in Databases Such As postgres and gaussdb

NOTE

Impact of rule violation:

- If the code or the compatibility setting of the built-in databases does not meet service requirements, you may need to migrate your data again.
- The time for changes to be applied may be prolonged if all services use built-in databases.

Solution:

- To meet the specific requirements of each service, it is recommended to create a dedicated database and allocate it accordingly.

Rule 2.2: Selecting the Suitable Database Code During Database Creation

NOTE

Impact of rule violation:

- Selecting the wrong database code may result in displaying garbled characters, and it is not possible to directly change the database code. In such cases, you will need to create a database and import the data again.

Solution:

- It is advisable to set the **ENCODING** to the UTF-8 format during database creation, unless there are specific requirements for a different encoding.

Rule 2.3: Choosing the Right Database Type for Compatibility with the Database to Be Created

NOTE

Impact of rule violation:

- Selecting the wrong type can lead to behavior inconsistencies after migrating the database from a different vendor to GaussDB(DWS). Unfortunately, it is not possible to directly change the compatibility setting. The only solution is to create a database and import the data again.

Solution:

- GaussDB(DWS) supports compatibility with databases like Teradata, Oracle, and MySQL. You can specify **DBCOMPATIBILITY** to set the compatible database type when creating a database.

Suggestion 2.4: Storing Objects with Associated Calculations in the Same Database

NOTE

Impact of rule violation:

- Cross-database access tends to have poorer performance compared to performing operations within the same database.

Solution:

- If multiple databases are created, it is advisable to store objects requiring associated calculations in the same database.

2.3.2 USER Object Design

Rule 2.5: Following the Least Privilege Principle and Avoiding Running Services Using Users with Special Permissions

NOTE

Impact of rule violation:

- Super users and administrators have full access to a lot of things in the system and using these users to run services can pose security and control risks.

Solution:

- It is advised to use common users for service running, reserving users with special permissions for management operations.

Rule 2.6: Avoiding the Use of a Single Database Account for All Services

NOTE

Impact of rule violation:

- Cross-database access typically has lower performance compared to accessing operations within the same database.

Solution:

- Create administrators , service operation users, and O&M users for different purposes.
- Use different users to run different services for improved management and allocation of services and resources.

2.3.3 Schema Object Design

Suggestion 2.7: Avoiding the Creation of Objects Under Other Users' Private Schemas

NOTE

A private schema refers to a schema with the same name as the user when the user is created. This schema is only accessible to the user.

Impact of rule violation:

- When you create an object under someone else's private schema, the permissions for that object are determined by the schema owner.

Solution:

- Create objects under your own private schema to have full control over the object permissions.

2.3.4 TABLESPACE Object Design

Rule 2.8 Avoiding Tablespace Customization

NOTE

Impact of rule violation:

- In a distributed scenario, using a custom tablespace to create a table can result in the table data not being stored in a distributed manner by DN, leading to storage skew.

Solution:

- Use the built-in default tablespace when creating a table object.

2.3.5 TABLE Object Design (Prioritized)

Rule 2.9: Selecting the Optimal Distribution Method and Columns During Table Creation

NOTE

Impact of rule violation:

- Incorrect distribution method and column selection can cause storage skew, deteriorate access performance, and even overload storage and computing resources.

Solution:

- When creating a table, it is important to use the **DISTRIBUTE BY** clause to explicitly specify the distribution method and distribution columns. The table below provides principles for selecting the distribution columns.

Table 2-2 Distribution column selection

Distribution Method	Description	Scenario
Hash	Table data is distributed to each DN based on the mapping between hash values generated by distribution columns and DNs. ● **Advantage**: Each DN contains only part of data, which is space-saving. ● **Disadvantage**: The even distribution of data depends heavily on the selection of distribution columns. If the join condition does not include the distribution columns of each node, data communication between nodes will be required.	Large tables and fact tables

Distribution Method	Description	Scenario
RoundRobin	Table data is distributed to DNs in polling mode. ● **Advantage**: Each DN only contains a portion of the data, taking up a small amount of space. Data is evenly distributed in polling mode and does not rely on distribution columns, eliminating data skews. ● **Disadvantage**: Using distribution column conditions cannot eliminate or reduce inter-node communication. In this scenario, the performance is inferior to that of **HASH**.	Large tables, fact tables, and tables without proper distribution columns
Replication	Full data in a table is copied to each DN in the cluster. ● **Advantage**: Each DN holds the complete data of the table. The **JOIN** operation avoids data communication between nodes, reducing network overhead and the overhead of starting and stopping the **STREAM** thread. ● **Disadvantage**: Each DN retains complete table data, which is redundant and occupies more storage space.	Small tables and dimension tables

Rule 2.10 Selecting an Optimal Storage Type During Table Creation

NOTE

Impact of rule violation:

- Row-store tables are not properly used. As a result, the query performance is poor and resources are overloaded.
- Improper use of column-store tables causes CU expansion, poor performance, and resource overload.

Solution:

- When creating a table, use **orientation** to explicitly specify the storage type. The following table describes the rules for selecting a storage type.

Table 2-3 Storage type selection

Storage Type	Applicable Scenario	Inapplicable Scenario
Row storage	● DML addition, deletion, and modification: scenarios with many **UPDATE** and **DELETE** operations ● DML query: point query (simple index–based query that returns only a few records)	DML query: statistical analysis query (with mass data involved in **GROUP** and **JOIN** processes) **CAUTION** When creating a row-store table (**orientation** is set to **row**), do not specify the **compress** attribute or use a row-store compressed table.
Column storage	● DML addition, deletion, and modification: **INSERT** batch import scenario (The number of data records imported to a single partition at a time is approximately 60,000 times the number of DNs or greater.) ● DML query: statistical analysis query (with mass data involved in **GROUP** and **JOIN** processes)	● DML addition, deletion, and modification: scenarios with many **UPDATE**/**DELETE** operations or a small number of **INSERT** operations ● DML query: high-concurrency point query

Rule 2.11 Selecting an Optimal Partitioning Policy During Table Creation

NOTE

Impact of rule violation:

Without partitioning, query performance and data governance efficiency will deteriorate. The larger the data volume, the greater the deterioration. The advantages of partitioning include:

- High query performance: The system queries only the concerned partitions rather than the whole table, improving the query efficiency.
- Improved data governance efficiency: In the data lifecycle management scenario, performing **TRUNCATE** or **DROP PARTITION** on historical partitions is much more efficient and effective than using **DELETE**.

Solution:

- Design partitions for tables that contain fields of the time type.

Table 2-4 Partitioning policy selection

Partitioning Policy	Description	Scenario
Range partitioning	Data is stored in different partitions based on the range of partition key values. The partition key ranges are consecutive but not overlapped.	1. The date or time field is used as the partition key. 2. Most queries contain partition keys as filter criteria. 3. Periodically delete data based on the partition key.
List partitioning	Partitioning is performed based on a unique list of partition key values.	1. A specific number of enumerated values are used as partition key values. 2. Most queries contain partition keys as filter criteria.

Suggestion 2.12: Designing Table Columns for Fast and Accurate Queries

NOTE

Impact of rule violation:

- The system may have limited storage space and low query efficiency.

Solution:

1. Design the table columns well for fast queries.
   - If possible, use integers instead of floating points or characters.
   - When using variable-length character type, specify the maximum length based on data features.
2. Design the table columns well for accurate queries.
   - Use the time type instead of the character type to store time data.
   - Use the minimum numeric type that meets the requirements. Avoid using bigint if int or smallint is sufficient to save space.
3. **Correctly use the constraints.**
   - Add **NOT NULL** constraints to columns that never have NULL values. The optimizer automatically optimizes the columns in certain scenarios.
   - Do not use the **DEFAULT** constraint for fields that can be supplemented at the service layer. Otherwise, unexpected results may be generated during data loading.
4. **Avoid unnecessary data type conversion.**
   - In tables that are logically related, columns having the same meaning should use the same data type.
   - Different types of comparison operations cause data type conversion, which may cause index and partition pruning failures and affect query performance.

Suggestion 2.13: Avoiding the Usage of Auto-increment Columns or Data Types

NOTE

Impact of rule violation:

- When auto-increment sequences or data types are heavily used, the GTM may become overloaded and slow down sequence generation.

Solution:

- Set a UUID to obtain a unique ID.
- If the auto-increment sequence must be used and there is no strict requirement for increasing order, you can set the cache, for example, **1000**, to reduce the pressure on GTM.

2.3.6 INDEX Object Design (Prioritized)

Rule 2.14: Creating Necessary Indexes and Selecting Optimal Columns and Sequences for Them

NOTE

Impact of rule violation:

- Redundant indexes consume unnecessary space and can impact data import efficiency.
- The column sequence in the composite index is incorrect, affecting the query efficiency.

Best practices:

The following conditions must be met when indexes are used:

- The index column should be a column commonly used for filtering or joining conditions.
- The index column should have more distinct values.
- When creating a multi-column combination index, prioritize columns with more distinct values.
- The number of indexes in a single table should be limited to less than five. You can control the number of indexes by combining them.
- In scenarios where data is added, deleted, or modified in batches, delete the index first and then add it back after the batch operation is complete to improve performance (real-time access may be affected).

Suggestion 2.15: Optimizing Performance by Choosing the Right Index Type and Avoiding Indexes for Column-Store Tables

NOTE

Impact of rule violation:

- Incorrect indexes do not improve column-store access and can negatively affect query performance.

Solution:

1. Specify the appropriate index type when creating indexes, avoiding the default psort index.
2. In point queries where small amounts of data need to be retrieved from mass datasets, consider creating a B-tree index.
3. For high range query performance, create a partial cluster key (PCK) to quickly filter and scan fact tables using the min/max sparse index. 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.
   - [Suggestion] Create a PCK for the filter condition column of the expression (e.g., **col op const**, where **op** is the operator =, >, >=, <=, and <, and **const** is a constant value).

2.3.7 VIEW Object Design

Suggestion 2.16: Limiting View Nesting to Three Layers

NOTE

Impact of rule violation:

- Too many nested views can lead to unstable execution plans and unpredictable time consumption.
- The risk of rebuilding objects on which views depend is high and the probability of lock conflicts increases.

Solution:

- Create views based on physical tables.

2.4 GaussDB(DWS) SQL Statement Development Specifications

2.4.1 DDL Operations

Suggestion 3.1: Avoiding Performing DDL Operations (Except CREATE) During Peak Hours or in Long Transactions

NOTE

Impact of rule violation:

DDL operations like **ALTER**, **DROP**, **TRUNCATE**, **REINDEX**, and **VACUUM FULL** have high lock levels and can block services during execution.

- During peak hours, these DDL operations with high lock levels should be avoided to prevent service blockage.
- Long transactions involving DDL operations with held or waited locks can also block services.

Solution:

- Choose off-peak hours or maintenance windows for DDL operations based on service periods. Specify the DDL execution environment and time consumption to avoid service blockage due to long lock waiting duration.

Rule 3.2: Specifying the Scope of Objects to Be Deleted When Using DROP

DANGER

Impact of rule violation:

Be cautious when using **DROP OBJECT** (e.g., **DATABASE**, **USER/ROLE**, **SCHEMA**, **TABLE**, **VIEW**) as it may cause data loss, especially with **CASCADE** deletions.

- **DROP DATABASE**: deletes all objects in the database.
- **DROP USER**: deletes the **USER** object and its schemas and table objects.
- **DROP SCHEMA**: deletes all objects in the schema.
- **DROP TABLE**: deletes the **TABLE** object and the indexes and views that depend on it.

Solution:

- Exercise caution when performing the **DROP** operation and back up data in advance.

2.4.2 INSERT Operation

Rule 3.3: Replacing INSERT with COPY for Efficient Multi-Value Batch Insertion

NOTE

Impact of rule violation:

- Parsing multiple values is time-consuming and resource-intensive, leading to low efficiency when importing data into the database.

Solution:

- Instead of using **INSERT VALUES**, the frontend should use APIs like CopyManager of JDBC.

Suggestion 3.4: Avoiding Performing Real-time INSERT Operations on Common Column-store Tables

NOTE

Impact of rule violation:

- Importing a small batch of data in real-time to a common column-store table can significantly expand the small CU, occupying a lot of storage space and impacting the query performance.

Solution:

- In real-time **INSERT** scenarios, evaluate the amount of data to be imported at once and the total amount of data. If the total amount of data is small, use row-store tables.
- In the real-time INSERT scenario, import around 60,000 data records to a single table, partition, or DN at a time. The minimum import batch is 5,000 data records.
- In the real-time **INSERT** scenario, use H-Store column-store tables (for version 8.3.0 or later).

2.4.3 UPDATE and DELETE Operations

Suggestion 3.5: Preventing Simultaneous Updates or Deletions of the Same Row in a Row-store Table

NOTE

Impact of rule violation:

- Concurrent **UPDATE** and **DELETE** operations on row-store tables may cause row lock blockage and distributed deadlocks, which can lead to service errors and performance degradation.

Solution:

- Group **UPDATE** and **DELETE** operations by primary key or distribution column. Perform parallel operations between groups while keeping operations within a group serial.

Suggestion 3.6: Avoiding Frequent or Simultaneous UPDATE and DELETE Operations on Column-store Tables

NOTE

Impact of rule violation:

- Frequent **UPDATE** and **DELETE** operations on column-store tables can result in CU bloat, leading to large space occupation and decreased access performance.
- Concurrent **UPDATE** and **DELETE** operations on row-store tables may cause row lock blockage and distributed deadlocks, which can lead to service errors and performance degradation.

Solution:

- Design tables with frequent **UPDATE** and **DELETE** operations as row-store tables.
- Group **UPDATE** and **DELETE** operations by primary key or distribution column. Perform parallel operations between groups while keeping operations within a group serial.

2.4.4 SELECT Operation

Rule 3.7: Avoiding Executing SQL Statements That Do Not Support Pushdown

NOTE

GaussDB(DWS) uses a distributed architecture, and to achieve optimal performance, SQL statements need to be pushed down to utilize distributed computing resources.

Impact of rule violation:

- SQL statements that are not pushed down may experience poor execution performance and, in severe cases, can lead to CN resource bottlenecks, impacting overall services.

Solution:

- Do not use syntax or functions that cannot be executed near the data source. For details, see **Optimizing Statement Pushdown**.

Rule 3.8: Specifying Association Conditions when Multiple Tables Are Associated

NOTE

Impact of rule violation:

- If no association condition is specified when linking multiple tables, it will result in a Cartesian product calculation. This can lead to an expanded result set, posing risks of performance issues and resource overload.

Solution:

- Specify filter and association conditions for each table during the association process.

Rule 3.9: Ensuring Consistency of Data Types in Associated Fields across Multiple Tables

NOTE

Impact of rule violation:

- Ensure consistent data types for associated fields to avoid unnecessary type conversions, data redistribution issues, and hindered generation of optimal plans.

Solution:

- Use the same data type for associated fields when tables are associated.

Suggestion 3.10: Avoiding Function Calculation on Association and Filter Condition Fields

NOTE

Impact of rule violation:

- In cases where function calculations are involved in association and filter conditions, the optimizer may fail to obtain accurate field statistics, impacting execution performance.

Solution:

- When comparing association condition fields, process the data before importing it into the database, especially when calculations are required for comparison.
- When filter criteria are compared with constants, perform function calculation only on constant columns. The following is an example:
```
SELECT id, from_image_id, from_person_id, from_video_id
FROM face_data
WHERE SS.DEL_FLAG = 'N'
AND NVL(SS.DELETE_FLAG, 'N') = 'N'
The modification is as follows:
SELECT id, from_image_id, from_person_id, from_video_id
FROM face_data
where SS.DEL_FLAG = 'N'
AND (SS.DELETE_FLAG = 'N' or SS.DELETE_FLAG is null)
```

Suggestion 3.11: Performing Pressure Tests and Concurrency Control for Resource-intensive SQL Statements

NOTE

Impact of rule violation:

- Storage and computing resources are overloaded, and the overall running performance deteriorates.

Solution:

A resource-intensive SQL statement contains:

- A large number of **UNION ALL**.
- A large number of AGGs (such as **COUNT DISTINCT** and **MAX**).
- A lot of **JOIN** operations for a large number of tables.
- A large number of **STREAM** operators (plan dimension).

Before rolling out, conduct pressure tests and implement concurrency control for certain SQL statements. If the resource capacity is exceeded, optimizing the service should be prioritized before reassessing the rollout plan.

Rule 3.12: Avoiding Excessive COUNT Operations on Large Row-store Tables

NOTE

If SSDs or other high-performance disk types are used, it may not be necessary to adhere strictly to this rule, but it is still crucial to monitor the I/O consumption.

Impact of rule violation:

- Performing frequent **COUNT** operations on large row-store tables can consume a significant amount of I/O resources, potentially leading to performance issues if an I/O bottleneck occurs.

Solution:

- Reduce the frequency of **COUNT** operations, use result caching, and collect statistics by partition to minimize I/O consumption.

Suggestion 3.13: Avoid Getting Large Result Sets (Except for Data Exports)

NOTE

Impact of rule violation:

- If you do not need to view all the results, querying ultra-large result sets becomes inefficient and wasteful in terms of resources.

Solution:

- Use the **LIMIT** clause to retrieve only the necessary result segments.
- Use a cursor to obtain the result sets by segment and set an appropriate value for **FETCH SIZE** if you need to query a large number of result sets.

Suggestion 3.14: Avoiding the Usage of SELECT * for Queries

NOTE

Impact of rule violation:

- Querying unnecessary columns increases the computing load and wastes computing resources.

Solution:

- Clearly list the fields required for the query in the **SELECT** statement to improve the query performance.

Suggestion 3.15: Using WITH RECURSIVE with Defined Termination Condition for Recursion

NOTE

Impact of rule violation:

- In cases where there is no specific termination condition, recursive operations can enter an infinite loop.
- Recursive operations generate duplicate data and occupy excessive resources.

Solution:

- Design proper termination conditions based on the volume and characteristics of the data in the service table.

Suggestion 3.16: Setting Schema Prefix for Table and Function Access

NOTE

Impact of rule violation:

- If the schema name prefix is not specified, the search will be performed sequentially across all tablespaces based on the tablespace list in the current **search_path**. This can lead to accessing unexpected tables due to schema switchover.

Solution:

- To enhance readability, stability, and portability, explicitly specify the schema prefix as **SCHEMA.** when accessing tables and function objects.

Suggestion 3.17: Identifying an SQL Statement with a Unique SQL Comment

NOTE

Impact of rule violation:

- The service's source tracing capability is limited. You can only verify it with R&D engineers using the database, user name, and client IP address.

Solution:

- You are advised to use **query_band**. The following is an example:
```
SET query_band='JobName=abc;AppName=test;UserName=user';
```
- Add a unique comment for each SQL statement to facilitate troubleshooting and application performance analysis. The following is an example of such comment.
```
/* Module name_Tool name_Job name_Step */, for example, /* mca_python_xxxxxx_step1 */ insert
into xxx select … from
```

2.5 GaussDB(DWS) Stored Procedure Development Specifications

Suggestion 4.1: Simplifying Stored Procedures and Avoiding Nesting

NOTE

Impact of rule violation:

- The maintenance cost for complex and nested stored procedures is high, making fault locating and recovery time-consuming.

Solution:

- Avoid using stored procedures altogether or limit their usage to a single layer. Nested stored procedures should be avoided.
- Create a corresponding log table for the stored procedure design and record information before and after key steps in the log table. Follow the steps below to implement this.

Saving and Viewing Logs

Step 1 Create a log table.

```
CREATE TABLE func_exec_log
(
id varchar2(32) default lower(sys_guid()),
pro_name varchar2(60),
exec_times int,
log_date date,
deal_date date,
log_mesage text
);
```

Step 2 Create a table and import data.

```
CREATE TABLE demo_table(data_id int, data_number int);
INSERT INTO demo_table values(generate_series(1,1000),generate_series(1,1000));
```

Step 3 Create a service stored procedure.

```
CREATE OR REPLACE FUNCTION demo_table_process(out exe_info text)
LANGUAGE plpgsql
AS $$
declare v_count int;
pro_result text;
fun_name   text;
exec_times   int;
begin
fun_name := 'demo_table_process';
```

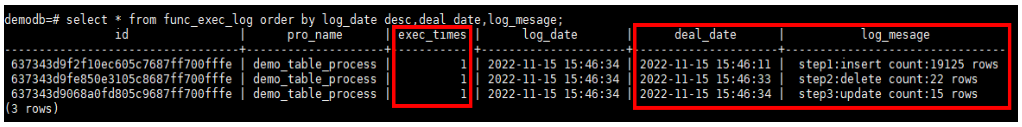
```
select nvl(max(exec_times), '0') + 1 into exec_times from func_exec_log where pro_name = fun_name;
-- Insert data into the service table.
insert into demo_table values (dbms_random.value(1, 1000)::int,generate_series(1,
dbms_random.value(10000, 20000)::int));
get diagnostics v_count = ROW_COUNT;
exe_info = sysdate || '# step1:insert count:' || v_count || ' rows;';
-- Delete specified data from a service table.
delete from demo_table where data_id = dbms_random.value(1, 1000)::int;
get diagnostics v_count = ROW_COUNT;
exe_info = exe_info || sysdate || '# step2:delete count:' || v_count || ' rows;';
-- Update service table data.
update demo_table set data_number = dbms_random.value(1, 100)::int where data_id =
dbms_random.value(1, 1000)::int;
exe_info = exe_info || sysdate || '# step3:update count:' || sql%rowcount || ' rows';
-- Record logs either before the entire program ends or after each step completes. You can also create a
function specifically for logging purposes.
insert into func_exec_log(pro_name, exec_times, log_date, deal_date, log_mesage) values
(fun_name,exec_times,sysdate,split_part(regexp_split_to_table(exe_info, ';'), '#',
1),split_part(regexp_split_to_table(exe_info, ';'), '#', 2));
-- EXCEPTION is used to ensure that logs can be properly recorded when the insertion, update, or deletion
exits abnormally.
EXCEPTION
WHEN OTHERS THEN
pro_result := exe_info || sysdate || '# exception error message is: ' || sqlerrm;
insert into func_exec_log(pro_name, exec_times, log_date, deal_date, log_mesage)
values(fun_name,exec_times,sysdate,split_part(regexp_split_to_table(pro_result, ';'), '#',
1),split_part(regexp_split_to_table(pro_result, ';'), '#', 2));
END; $$;
```

Step 4 Invoke the stored procedure (normal execution).

```
SELECT demo_table_process();
```

Step 5 View the created log table to check the service running status.

```
SELECT * FROM func_exec_log ORDER BY log_date desc,deal_date,log_mesage;
```



```
demodb=# select * from func_exec_log order by log_date desc,deal_date,log_mesage;
                id                |      pro_name      | exec_times |      log_date       |      deal_date      |           log_mesage
----------------------------------+--------------------+------------+---------------------+---------------------+---------------------------------
 637343d9f2f10ec605c7687ff700fffe | demo_table_process |          1 | 2022-11-15 15:46:34 | 2022-11-15 15:46:11 |  step1:insert count:19125 rows
 637343d9fe850e3105c8687ff700fffe | demo_table_process |          1 | 2022-11-15 15:46:34 | 2022-11-15 15:46:33 |  step2:delete count:22 rows
 637343d9068a0fd805c9687ff700fffe | demo_table_process |          1 | 2022-11-15 15:46:34 | 2022-11-15 15:46:34 |  step3:update count:15 rows
(3 rows)
```

Step 6 Invoke the stored procedure again to construct an execution exception.

```
SELECT demo_table_process();  -- Delete the data_number column of demo_table to construct an exception,
and then call the stored procedure again.
```

Step 7 View the log to check the service running status.

----End

Rule 4.2: Avoiding Non-CREATE DDL Operations in Stored Procedures

NOTE

Impact of rule violation:

- A stored procedure is a large transaction. If a non-CREATE DDL operation, especially one with a high lock level, is executed, it can block external access to related tables during the stored procedure's execution window.

Solution:

- Avoid using non-CREATE DDL operations within stored procedures whenever possible. If there is a necessity to use such operations, carefully assess the duration of the stored procedures and the potential impact of the DDL operations. It is advised to schedule non-CREATE DDL operations during off-peak hours when external access services are less active.

2.6 Detailed Design Rules for GaussDB(DWS) Objects

2.6.1 GaussDB(DWS) Database Object Naming Rules

The name of a database object must contain 1 to 63 characters, start with a letter or underscore (_), and can contain letters, digits, underscores (_), and dollar signs ($).

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

  NOTE

  You can run **SELECT * FROM pg_get_keywords()** to query GaussDB(DWS) keywords or view the keywords in section "Keywords" in *SQL Syntax Reference*.
- [Proposal] Do not use strings enclosed in double quotation marks to define database object names. In GaussDB(DWS), double quotation marks are used to specify that the enclosed database object names are case sensitive. 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.
  - A database object name consists of letters, digits, and underscores (_); and cannot start with a digit. You are advised to use multiple words separated with hyphens (-).
  - 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 indicating actual business terms. The naming format should be consistent within a cluster.
  - 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 data set.
  - 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*.

2.6.2 GaussDB(DWS) Database Object Design Rules

2.6.2.1 GaussDB(DWS) Database and Schema Design Rules

In GaussDB(DWS), 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 Suggestions

- Create databases as required. Do not use the default **gaussdb** database of a cluster.
- Create a maximum of three user-defined databases in a cluster.
- To make your database encoding compatible with most characters, you are advised to use the UTF-8 encoding when creating a database.
- Exercise caution when you set **ENCODING** and **DBCOMPATIBILITY** configuration items during database creation. In GaussDB(DWS), **DBCOMPATIBILITY** can be set to **TD**, **Oracle**, or **MySQL** to be compatible with Teradata, Oracle, or MySQL syntax, respectively. Syntax behavior may vary with the three modes. For details, see **Syntax Compatibility Differences Among Oracle, Teradata, and MySQL**.
- By default, a database owner has all permissions for all objects in the database, including the deletion permission. Exercise caution when using the deletion permission.

Schema Design Suggestions

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

2.6.2.2 GaussDB(DWS) Table Design Rules

GaussDB(DWS) uses a distributed architecture. Data is distributed on DNs. Comply with the following principles to properly design a table:

- [Notice] Evenly distribute data on each DN to prevent data skew. If most data is stored on several DNs, the effective capacity of a cluster decreases. Select a proper distribution column to avoid data skew.
- [Notice] Evenly scan each DN when querying tables. Otherwise, DNs most frequently scanned will become the performance bottleneck. For example, when you use equivalent filter conditions on a fact table, the nodes are not evenly scanned.
- [Notice] Reduce the amount of data to be scanned. You can use the pruning mechanism of a partitioned table.

- [Notice] Minimize random I/O. 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.
- [Notice] Try to avoid data shuffling. To shuffle data is to physically transfer it from one node to another. This unnecessarily occupies many network resources. To reduce network pressure, locally process data, and to improve cluster performance and concurrency, you can minimize data shuffling by using proper association and grouping conditions.

Selecting a Storage Mode

[Proposal] Selecting a storage mode is the first step in defining a table. The storage mode mainly depends on the user's service type. For details, see **Table 2-5**.

Table 2-5 Table storage modes and scenarios

Storage Mode	Benefit	Drawback	Application Scenarios
Row storage	Data is stored by row. When you query a row of data, you can quickly locate the target row.	All data in the queried row is read while only a few columns are needed.	1. The number of columns in the table is small, and most fields in the table are queried. 2. Point queries (simple index–based query that returns only a few records) are performed. 3. Add, Delete, Modify, and Query operations on entire rows are frequently performed.
Column storage	1. Only necessary columns in a query are read. 2. The homogeneity of data within a column facilitates efficient compression.	It is not suitable for INSERT or UPDATE operations on a small amount of data.	1. Query a few columns in a table that contains a large number of columns. 2. Statistical analysis queries (requiring a large number of association and grouping operations) 3. Ad hoc queries (using uncertain query conditions and unable to utilize indexes to scan row-store tables)

Selecting a Distribution Mode

[Proposal] Comply with the following rules to distribute table data.

Table 2-6 Table distribution modes and scenarios

Distribution Mode	Description	Application Scenarios
Hash	Table data is distributed on all DNs in a cluster by hash.	Fact tables containing a large amount of data
Replication	Full data in a table is stored on every DN in a cluster.	Dimension tables and fact tables containing a small amount of data
Round-robin	Each row of the table is sent to each DN in turn. Therefore, data is evenly distributed on each DN.	Fact tables that contain a large amount of data and cannot find a proper distribution column in hash mode

Selecting a Partitioning Mode

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

- [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.

The example of a partitioned table definition is as follows:

```
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)
);
```

Selecting a Distribution Key

Selecting a distribution key is important for a hash table. An improper distribution key may cause data skew. As a result, the I/O load is heavy on several DNs, affecting the overall query performance. After you select a distribution policy for a hash table, check for data skew to ensure that data is evenly distributed. Comply with the following rules to select a distribution key:

- [Proposal] Select a column containing discrete data as the distribution key, so that data can be evenly distributed on each DN. If a single column is not discrete enough, consider using multiple columns as distribution keys. You can select the primary key of a table as the distribution key. For example, in an employee information table, select the certificate number column as the distribution key.
- [Proposal] If the first rule is met, do not select a column having constant filter conditions as the distribution key. For example, in a query on the **dwcjk** table, if the **zqdh** column contains the constant filter condition **zqdh='000001'**, avoid selecting the **zqdh** column as the distribution key.
- [Proposal] If the first and second rules are met, select the join conditions in a query as distribution keys. If a join condition is used as a distribution key, the data involved in a join task is locally distributed on DNs, which greatly reduces the data flow cost among DNs.

2.6.2.3 GaussDB(DWS) Column Design Rules

Selecting a Data Type

Comply with the following rules to improve query efficiency when you design 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(DWS).

Table 2-7 Common string types

Parameter	**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

Parameter	Description	Max. Storage Capacity
TEXT	Variable-length string. Its maximum length is 8203 bytes less than 1 GB.	8203 bytes less than 1 GB

2.6.2.4 GaussDB(DWS) Constraint Design Rules

DEFAULT and NULL Constraints

- [Proposal] If all the column values can be obtained from services, you are not advised to use the **DEFAULT** constraint, because doing so will generate unexpected results 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 Key

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(DWS). 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 least distinct values.

Unique Constraint

- [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 Constraint

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

Check Constraint

- [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*.

2.6.2.5 Design Rules for GaussDB(DWS) Views and Associated Tables

View Design

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

Joined Table Design

- [Proposal] Minimize joined columns across tables.
- [Proposal] Joined columns should use the same data type.
- [Proposal] The names of associated fields should show the associations. For example, they can use the same name.

2.6.3 GaussDB(DWS) SQL Writing Rules

DDL

- [Proposal] In GaussDB(DWS), 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(DWS) 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 use **DROP... CASCADE** to delete objects unless the dependencies between objects are specified. Otherwise, objects may be deleted by mistake.

Data Loading and Uninstalling

- [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] Perform type coercion to convert data types. 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] In Oracle compatibility mode, null strings will be automatically converted to NULL during data import. If a null string needs to be reserved, you need to create a database that is compatible with Teradata.

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 *****. 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] Do not use 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 with 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.
- [Notice] When constants are used to filter data, the system searches for functions used for calculating these two data types based on the data types of the constants and matched columns. If no function is found, the system converts the data type implicitly. Then, the system searches for a function used for calculating the converted data type.

  ```
  SELECT * FROM test WHERE timestamp_col = 20000101;
  ```

  In the preceding example, if **timestamp_col** is the timestamp type, the system first searches for the function that supports the "equal" operation of the timestamp and int types (constant numbers are considered as the int type). If no such function is found, the **timestamp_col** data and constant numbers are implicitly converted into the text type for calculation.
- [Proposal] Do not use scalar subquery statements. A scalar subquery appears in the output list of a **SELECT** statement. In the following example, the part enclosed in parentheses 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 filtering conditions should be sorted. The condition that few records are selected for reading (the number of filtered records is small) is listed at the beginning.
- [Proposal] The filter criteria in the WHERE clause should comply with the unilateral rule. That is, the field name is placed on one side of the comparison condition. This allows the optimizer to automatically perform pruning optimization in some scenarios. Filtering 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. For 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 sorting operations. Sorting requires a large amount of memory and CPU. If service logic permits, **ORDER BY** and **LIMIT** can be combined to reduce resource overhead. By default, data in GaussDB(DWS) is sorted by ASC & NULL LAST.
- [Proposal] When the **ORDER BY** clause is used for sorting, specify sorting 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 filtering 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. For example:

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

- [Proposal] If an **IN(val1, val2, va...)** expression contains a large number of columns, you are advised to replace it with the **IN (values (va1), (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 T1,T2 WHERE T1.C1=T2.C2);
```

NOTE

- If you cannot ensure that the values of the **T1.C1** column are **NOT NULL**, you cannot use **(NOT) EXIST** instead of **(NOT) IN**.
- If T1.C1 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 it and commit transaction once the query is finished.

2.6.4 GaussDB(DWS) JDBC Configuration Rules

Currently, third-party tools are connected to GaussDB(DWS) trough JDBC. This section describes the precautions for configuring the tools.

Connection Parameters

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

```
params = {
{ "user", user },
{ "database", database },
{ "client_encoding", "UTF8" },
{ "DateStyle", "ISO" },
{ "extra_float_digits", "2" },
{ "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 connecting to the database through JDBC, ensure that the following two time zones are the same:
  - Time zone of the host where the JDBC client is located
  - Time zone of the host where the GaussDB(DWS) server is located

fetchsize

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

autocommit

[Proposal] It is recommended that you enable the **autocommit** switch in the code for connecting to GaussDB(DWS) 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(DWS) to release occupied resources. You are advised to set the session timeout interval in the task.

[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, before you return the connection to the connection pool, run **SET SESSION AUTHORIZATION DEFAULT;RESET ALL;** to clear the connection status.
- If a temporary table is used, delete it 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(DWS) JDBC to import data in batches during application development.

For how to use CopyManager, see **CopyManager**.

2.6.5 Rules for Using Custom GaussDB(DWS) External Functions (pgSQL/Java)

- [Notice] Java UDFs can perform some Java logic calculation. Do not encapsulate services in Java UDFs.
- [Notice] Do not connect to a database in any way (for example, by using JDBC) in Java functions.
- [Notice] Only the data types listed in the following table can be used. User-defined types and complex data types (Java Array and derived classes) are not supported.
- [Notice] User-defined aggregation functions (UDAFs) and user-defined table-generating functions (UDTFs) are not supported.

Table 2-8 PL/Java mapping for default data types

GaussDB(DWS)	Java
BOOLEAN	boolean
"char"	byte
bytea	byte[]
SMALLINT	short
INTEGER	int
BIGINT	long
FLOAT4	float
FLOAT8	double
CHAR	java.lang.String
VARCHAR	java.lang.String
TEXT	java.lang.String
name	java.lang.String
DATE	java.sql.Timestamp
TIME	java.sql.Time (stored value treated as local time)
TIMETZ	java.sql.Time
TIMESTAMP	java.sql.Timestamp
TIMESTAMPTZ	java.sql.Timestamp

2.6.6 Rules for Using GaussDB(DWS) PL/pgSQL

General Principles

1. Development shall strictly comply with design documents.
2. Program modules shall be highly cohesive and loosely coupled.
3. Proper, comprehensive troubleshooting measures shall be developed.
4. Code shall be reasonable and clear.
5. Program names shall comply with a unified naming rule.
6. Fully consider the program efficiency, including the program execution efficiency and database query and storage efficiency. Use efficient and effective processing methods.
7. Program comments shall be detailed, correct, and standard.
8. The **COMMIT** or **ROLLBACK** operation shall be performed at the end of a stored procedure, unless otherwise required by applications.
9. Programs shall support 24/7 processing. In the case of an interruption, the applications shall provide secure, easy-to-use resuming features.
10. Application output shall be standard and simple. The output shall show the progress, error description, and execution results for application maintenance personnel, and provide clear and intuitive reports and documents for business personnel.

Programming Principles

1. Use bound variables in SQL statements in the PL/pgSQL.
2. **RETURNING** is recommended for SQL statements in PL/pgSQL.
3. Principles for using stored procedures:
   a. Do not use more than 50 output parameters of the Varchar or Varchar2 type in a stored procedure.
   b. Do not use the LONG type for input or output parameters.
   c. Use the CLOB type for output strings that exceed 10 MB.
4. Variable declaration principles:
   a. Use **%TYPE** to declare a variable that has the same meaning as that of a column or variable in an application table.
   b. Use **%ROWTYPE** to declare a record that has the same meaning as that of a row in an application table.
   c. Each line of a variable declaration shall contain only one statement.
   d. Do not declare variables of the LONG type.
5. Principles for using cursors:
   a. Explicit cursors shall be closed after being used.
   b. Cursor variables must be closed after being used. If a cursor variable needs to transfer data to the invoked application, close the cursor in the application. If a cursor variable is used only in a stored procedure, close the cursor explicitly.
   c. Before using **DBMS_SQL.CLOSE_CURSOR** to close a cursor, use **DBMS_SQL.IS_OPEN** to check whether the cursor is open.

6. Principles for collections: You are advised to use the FOR ALL statement instead of the FOR loop statement to reference elements in a collection.
7. Principles for using dynamic statements:
    a. Dynamic SQL shall not be used in the transaction programs of online systems.
    b. Dynamic SQL statements can be used to implement DDL statements and system control commands in PL/pgSQL.
    c. Variable binding is recommended.
8. Principles for assembling SQL statements:
    a. You are advised to use bound variables to assemble SQL statements.
    b. If the conditions for assembling SQL statements contain external input sources, the characters in the input conditions shall be checked to prevent attacks.
    c. In a PL/pgSQL script, the length of a single line of code cannot exceed 2499 characters.
9. Principles for using triggers:
    a. Triggers can be used to implement availability design in scenarios where differential data logs are irrelevant to service processing.
    b. Do not use triggers to implement service processing functions.

Exception Handling Principles

Any error that occurs in a PL/pgSQL function aborts the execution of the function and related transactions. You can use a **BEGIN** block with an **EXCEPTION** clause to catch and fix errors.

1. In a PL/pgSQL block, if an SQL statement cannot return a definite result, you are advised to handle exceptions (if any) in **EXCEPTION**. Otherwise, unhandled errors may be transferred to the external block and cause program logic errors.
2. You can directly use the exceptions that have been defined in the system. GaussDB(DWS) does not support custom exceptions.
3. 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.

Writing Standard

1. Variable naming rules:
    a. The input parameter format of a procedure or function is **IN_***Parameter_name*. The parameter name shall be in uppercase.
    b. The output parameter format of a procedure or function is **OUT_***Parameter_name*. The parameter name shall be in uppercase.
    c. The format for input and output parameters in a procedure or function is **IO_***Parameter name*, with the parameter name written in uppercase.
    d. When creating variables for procedures and functions, use the format **v_***Variable name*, with the variable name written in lowercase.

   e. In query concatenation, the concatenation variable name of the **WHERE** statement shall be **v_where**, and the concatenation variable name of the **SELECT** statement shall be **v_select**.
   f. The record type (TYPE) name shall consist of **T** and a variable name. The name shall be in uppercase.
   g. A cursor name shall consist of **CUR** and a variable name. The name shall be in uppercase.
   h. The name of a reference cursor (REF CURSOR) shall consist of **REF** and a variable name. The name shall be in uppercase.
2. Rules for defining variable types:
   a. Use **%TYPE** to declare the type of a variable that has the same meaning as that of a column in an application table.
   b. Use **%ROWTYPE** to declare the type of a record that has the same meaning as that of a row in an application table.
3. Rules for writing comments:
   a. Comments shall be meaningful and shall not just repeat the code content.
   b. Comments shall be concise and easy to understand.
   c. Comments shall be provided at the beginning of each stored procedure or function. The comments shall contain a brief function description, author, compilation date, program version number, and program change history. The format of the comments at the beginning of stored procedures shall be the same.
   d. Comments shall be provided next to the input and output parameters to describe the meaning of variables.
   e. Comments shall be provided at the beginning of each block or large branch to briefly describe the function of the block. If an algorithm is used, comments shall be provided to describe the purpose and result of the algorithm.
4. Variable declaration format:

   Each line shall contain only one statement. To assign initial values, write them in the same line.
5. Letter case:

   Use uppercase letters except for variable names.
6. Indentation:

   In the statements used for creating a stored procedure, the keywords **CREATE**, **AS/IS**, **BEGIN**, and **END** at the same level shall have the same indent.
7. Statement rules:
   a. For statements that define variables, Each line shall contain only one statement.
   b. The keywords **IF**, **ELSE IF**, **ELSE**, and **END** at the same level shall have the same indent.
   c. The keywords **CASE** and **END** shall have the same indent. The keywords **WHEN** and **ELSE** shall be indented.

d. The keywords **LOOP** and **END LOOP** at the same level shall have the same indent. Nested statements or statements at lower levels shall have more indent.

3 Creating and Managing GaussDB(DWS) Database Objects

3.1 Creating and Managing GaussDB(DWS) Databases

A database is a collection of objects such as tables, indexes, views, stored procedures, and operators. GaussDB (DWS) supports the creation of multiple databases. However, a client program can connect to and access only one database at a time, and cross-database query is not supported.

Template and Default Databases

- GaussDB (DWS) provides two template databases **template0** and **template1** and a default database gaussdb.
- By default, each newly created database is based on a template database. The GaussDB(DWS) database uses **template1** as the template by default. The encoding format is SQL_ASCII, and user-defined character encoding is not allowed. If you need to specify the character encoding when creating a database, use **template0** to create the database.
- Do not use a client or any other tools to connect to or to perform operations on both the two template databases.

NOTE

You can run the **show server_encoding** command to view the current database encoding.

Creating a Database.

Run the **CREATE DATABASE** statement to create a database.

```
CREATE DATABASE mydatabase;
```

NOTE

- When you create a database, if the length of the database name exceeds 63 bytes, the server truncates the database name and retains the first 63 bytes. Therefore, you are advised to set the length of the database name to a value less than or equal to 63 bytes. Do not use multi-byte characters as object names. If an object whose name is truncated mistakenly cannot be deleted, delete the object using the name before the truncation, or manually delete it from the corresponding system catalog on each node.
- Database names must comply with the naming convention of SQL identifiers. The current user 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 different schemas in the same database.
- A maximum of 128 databases can be created in GaussDB(DWS).
- You must have the permission to create a database or the permission that the system administrator owns.

Viewing Databases

To view databases, perform the following steps:

- Run the **\l** meta-command to view the database list of the database system.
```
\l
```
- Querying the database list using the **pg_database** system catalog
```
SELECT datname FROM pg_database;
```

Modifying a Database

You can use the **ALTER DATABASE** statement 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:
```
ALTER DATABASE mydatabase SET search_path TO pa_catalog,public;
```
- Rename the database.
```
ALTER DATABASE mydatabase RENAME TO newdatabase;
```

Deleting a Database

You can run **DROP DATABASE** statement to delete a database. This statement 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 being accessed by users cannot be deleted, You need to connect to another database before deleting this database.

Run the **DROP DATABASE** statement to delete a database:
```
DROP DATABASE newdatabase;
```

3.2 Creating and Managing GaussDB(DWS) Schemas

A schema is the logical organization of objects and data in a database. Schema management allows multiple users to use the same database without interfering with each other. Third-party applications can be added to corresponding schemas to avoid conflicts.

The same database object name can be used in different schemas in a 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 in a database.

If a user is created, a schema named after the user will also be created in the current database.

Public mode

Each database has a schema named **public**. All users have the ability to use the public schema in the database, but only certain roles have the authority to create objects within it.

Creating a Schema

- Run the **CREATE SCHEMA** command to create a schema.

  ```
  CREATE SCHEMA myschema;
  ```

  To create or access an object in the schema, the object name in the command should be composed of the schema name and the object name, which are separated by a dot (.), for example, **myschema.table**.

- Users can create a schema owned by others. For example, run the following command to create a schema named **myschema** and set the owner of the schema to user **jack**:

  ```
  CREATE SCHEMA myschema AUTHORIZATION jack;
  ```

  If **authorization username** is not specified, the schema owner is the user who runs the command.

Modifying a Schema

- Run the **ALTER SCHEMA** command to change the schema name. Only the schema owner can change the schema name.

  ```
  ALTER SCHEMA schema_name RENAME TO new_name;
  ```

- Run the **ALTER SCHEMA** command to change the schema owner.

  ```
  ALTER SCHEMA schema_name OWNER TO new_owner;
  ```

Setting the Schema Search Path

The GUC parameter **search_path** specifies the schema search sequence. The parameter value is a series of schema names separated by commas (,). If no schema is specified during object creation, the object will be added to the first schema displayed in the search path. If there are objects with the same name in 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.

- Run the **SHOW** command to view the current search path.

  ```
  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 to only to the **public** schema.

- Run the **SET** command to modify the default schema of the current session. For example, if the search path is set to "**myschema**, **public**", **myschema** is searched first.
  ```
  SET SEARCH_PATH TO myschema, public;
  ```
  You can also run the **ALTER ROLE** command to set search_path for a role (user). For example:
  ```
  ALTER ROLE jack SET search_path TO myschema, public;
  ```

Using 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 (.).

- Create a table **mytable** in **myschema**. Create a table in **schema_name.table_name** format.
  ```
  CREATE TABLE myschema.mytable(id int, name varchar(20));
  ```
- Query all data in the table **mytable** in **myschema**.
  ```
  SELECT * FROM myschema.mytable;
   id | name
  ----+------
  (0 rows)
  ```

Viewing a Schema

- Use the **current_schema()** function to view the current schema.
  ```
  SELECT current_schema();
   current_schema
  ----------------
   myschema
  (1 row)
  ```
- 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.
  ```
  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**.
  ```
  SELECT * FROM PG_NAMESPACE;
  ```
- Use the **PGXC_TOTAL_SCHEMA_INFO** view to query the space usage of schemas in the cluster.
  ```
  SELECT * FROM PGXC_TOTAL_SCHEMA_INFO;
  ```
- 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.
  ```
  SELECT distinct(tablename),schemaname FROM PG_TABLES where schemaname = 'pg_catalog';
  ```

Schema Permission Control

By default, a user can only access database objects in its own schema. To access objects in other schemas, the user must have the **usage** permission of the corresponding schema.

By granting the **CREATE** permission for a schema to a user, the user can create objects in this schema.

- Grant the **usage** permission of **myschema** to user **jack**.

```
GRANT USAGE ON schema myschema TO jack;
```

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

```
REVOKE USAGE ON schema myschema FROM jack;
```

Drop Schema

- Run the **DROP SCHEMA** command to delete an empty schema (no database objects in the schema).

```
DROP SCHEMA IF EXISTS myschema;
```

- By default, a schema must be empty before being deleted. To delete a schema and all its objects (such as tables, data, and functions), use the **CASCADE** keyword.

```
DROP SCHEMA myschema CASCADE;
```

System Schema

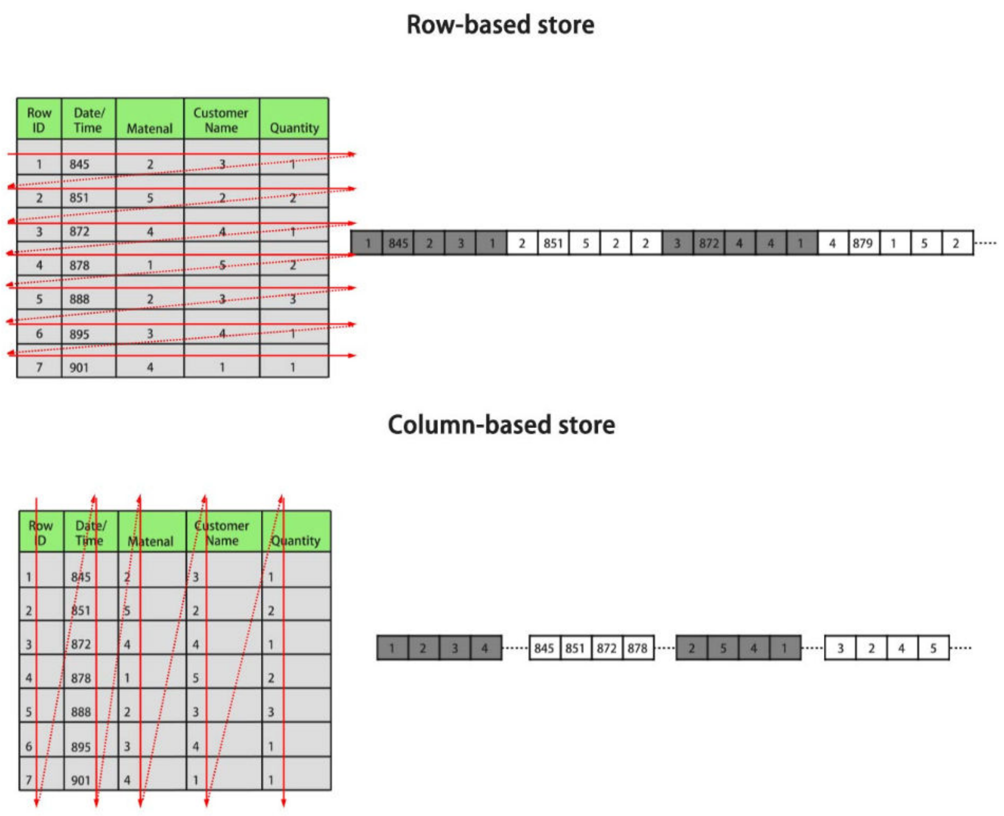
- Each database has a **pg_catalog** schema, which contains system catalogs and all built-in data types, functions, and operators. **pg_catalog** is a part of the search path and has the second highest search priority. It is searched after the schema of temporary tables and before other schemas specified 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.
- The **information_schema** consists of a collection of views that contain object information in a database. These views obtain system information from the system catalogs in a standardized way.

3.3 Selecting a GaussDB(DWS) Table Storage Model

GaussDB(DWS) supports hybrid row and column storage. When creating a table, you can set the table storage mode to row storage or column storage.

Row storage stores tables to disk partitions by row, and column storage stores tables to disk partitions by column. By default, a table is created in row storage mode. For details about differences between row storage and column storage, see **Figure 3-1**.

Figure 3-1 Differences between row storage and column storage



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.

The row/column storage of a table is specified by the **orientation** attribute in the table definition. The value **row** indicates a row-store table and **column** indicates a column-store table. The default value is **row**. Each storage mode applies to specific scenarios. Select an appropriate mode when creating a table.

Table 3-1 Table storage modes and scenarios

Storage Mode	Benefit	Drawback	Application Scenarios
Row storage	Data is stored by row. When you query a row of data, you can quickly locate the target row.	All data in the queried row is read while only a few columns are needed.	1. The number of columns in the table is small, and most fields in the table are queried. 2. Point queries (simple index–based query that returns only a few records) are performed. 3. Add, Delete, Modify, and Query operations on entire rows are frequently performed.
Column storage	1. Only necessary columns in a query are read. 2. The homogeneity of data within a column facilitates efficient compression.	It is not suitable for INSERT or UPDATE operations on a small amount of data.	1. Query a few columns in a table that contains a large number of columns. 2. Statistical analysis queries (requiring a large number of association and grouping operations) 3. Ad hoc queries (using uncertain query conditions and unable to utilize indexes to scan row-store tables)

Creating a Row-store Table

For example, to create a row-store table named **customer_t1**, run the following command:

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

Creating a column-store table.

For example, to create a column-store table named **customer_t2**, run the following command:

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

Table Compression

Table compression can be enabled when a table is created. Table compression enables data in the table to be stored in compressed format to reduce memory usage.

In scenarios where I/O is large (much data is read and written) and CPU is sufficient (little data is computed), select a high compression ratio. In scenarios where I/O is small and CPU is insufficient, select a low compression ratio. Based on this principle, you are advised to select different compression ratios and test and compare the results to select the optimal compression ratio as required. Specify a compressions ratio using the **COMPRESSION** parameter. The supported values are as follows:

- The valid value of column-store tables is **YES**, **NO**, **LOW**, **MIDDLE**, or **HIGH**, and the default value is **LOW**.
- The valid values of row-store tables are **YES** and **NO**, and the default is **NO**. (The row-store table compression function is not put into commercial use. To use this function, contact technical support.)

The service scenarios applicable to each compression level are described in the following table.

Compression Level	Application Scenario
LOW	The system CPU usage is high and the disk storage space is sufficient.
MIDDLE	The system CPU usage is moderate and the disk storage space is insufficient.
HIGH	The system CPU usage is low and the disk storage space is insufficient.

For example, to create a compressed column-store table named **customer_t3**, run the following command:

```
CREATE TABLE customer_t3
(
 state_ID   CHAR(2),
 state_NAME VARCHAR2(40),
 area_ID    NUMBER
)
WITH (ORIENTATION = COLUMN,COMPRESSION=middle);
```

3.4 Creating and Managing GaussDB(DWS) Tables

Creating a Table

You can run the **CREATE TABLE** command to create a table. When creating a table, you can define the following information:

- Columns and **data type** of the table.
- Table or column constraints that restrict a column or the data contained in a table. For details, see **Definition of Table Constraints**.
- Distribution policy of a table, which determines how the GaussDB (DWS) database divides data between segments. For details, see **Definition of Table Distribution**.
- Table storage format. For details, see **Selecting a GaussDB(DWS) Table Storage Model**.
- Partition table information. For details, see **Creating and Managing GaussDB(DWS) Partitioned Tables**.

Example: Use **CREATE TABLE** to create a table **web_returns_p1**, use **wr_item_sk** as the distribution key, and sets the range distribution function through **wr_returned_date_sk**.

```
CREATE TABLE web_returns_p1
(
    wr_returned_date_sk       integer,
    wr_returned_time_sk       integer,
    wr_item_sk                integer NOT NULL,
    wr_refunded_customer_sk   integer
)
WITH (orientation = column)
DISTRIBUTE BY HASH (wr_item_sk)
PARTITION BY RANGE(wr_returned_date_sk)
(
    PARTITION p2019 START(20191231) END(20221231) EVERY(10000),
    PARTITION p0 END(maxvalue)
);
```

Definition of Table Constraints

You can define constraints on columns and tables to restrict data in a table. However, there are the following restrictions:

- The primary key constraint and unique constraint in the table must contain a distribution column.
- Column-store tables support the **PARTIAL CLUSTER KEY** and table-level primary key and unique constraints, but do not support table-level foreign key constraints.
- Only the **NULL**, **NOT NULL**, and **DEFAULT** constant values can be used as column-store table column constraints.

Table 3-2 Table constraints

Constraint	Description	Example

Check constraint	A CHECK constraint allows you to specify that values in a specific column must satisfy a Boolean (true) expression.	Create the **products** table. The **price** column must be positive.
		`CREATE TABLE products`
		`(`
		`   product_no integer,`
		`   name text,`
		`   price numeric CHECK (price > 0)`
		`);`

NOT NULL constraint	A NOT NULL constraint specifies that a column cannot have null values. A non-null constraint is always written as a column constraint.	Create the **products** table. The values of **product_no** and **name** cannot be null.
		`CREATE TABLE products`
		`(`
		`    product_no integer NOT NULL,`
		`    name text NOT NULL,`
		`    price numeric`
		`);`

UNIQUE constraint	A UNIQUE constraint specifies that the values in a column or a group of columns are all unique. If **DISTRIBUTE BY REPLICATION** is not specified, the column table that contains only unique values must contain distribution columns.	Create the **products** table. The values of **product_no** must be unique.
		`CREATE TABLE products`
		`(`
		`    product_no integer UNIQUE,`
		`    name text,`
		`    price numeric`
		`)DISTRIBUTE BY HASH(product_no);`

Primary key constraint	A primary key constraint is the combination of a UNIQUE constraint and a NOT NULL constraint. If **DISTRIBUTE BY REPLICATION** is not specified, the column set with a primary key constraint must contain distributed columns. If a table has a primary key, the column (or group of columns) of the primary key is selected as the distribution keys of the table by default.	Create the **products** table. The primary key constraint is **product_no**.
		`CREATE TABLE products`
		`(`
		`    product_no integer PRIMARY KEY,`
		`    name text,`
		`    price numeric`
		`)DISTRIBUTE BY HASH(product_no);`

Constraint	Description	Example

Partial cluster key	Partial cluster key can minimize or maximize sparse indexes to quickly filter base tables. Partial cluster key can specify multiple columns, but you are advised to specify no more than two columns.	Create the **products** table with PCK set to **product_no**:
		`CREATE TABLE products`
		`(`
		`    product_no integer,`
		`    name text,`
		`    price numeric,`
		`    PARTIAL CLUSTER KEY(product_no)`
		`) WITH (ORIENTATION = COLUMN);`

Definition of Table Distribution

GaussDB(DWS) supports the following distribution modes: replication, hash, and roundrobin.

NOTE

The roundrobin distribution mode is supported only by cluster version 8.1.2 or later.

Policy	Description	Scenario	Advantages/Disadvantages
Replication	Full data in a table is stored on each DN in the cluster.	Small tables and dimension tables	● The advantage of replication is that each DN has full data of the table. During the join operation, data does not need to be redistributed, reducing network overheads and reducing plan segments (each plan segment starts a corresponding thread). ● The disadvantage of replication is that each DN retains the complete data of the table, resulting in data redundancy. Generally, replication is only used for small dimension tables.
Hash	Table data is distributed on all DNs in the cluster.	Fact tables containing a large amount of data	● The I/O resources of each node can be used during data read/write, greatly improving the read/write speed of a table. ● Generally, a large table (containing over 1 million records) is defined as a hash table.

Policy	Description	Scenario	Advantages/Disadvantages
Polling (Round-robin)	Each row in the table is sent to each DN in turn. Data can be evenly distributed on each DN.	Fact tables that contain a large amount of data and cannot find a proper distribution column in hash mode	● Round-robin can avoid data skew, improving the space utilization of the cluster. ● Round-robin does not support local DN optimization like a hash table does, and the query performance of Round-robin is usually lower than that of a hash table. ● If a proper distribution column can be found for a large table, use the hash distribution mode with better performance. Otherwise, define the table as a round-robin table.

Selecting a Distribution Key

If the hash distribution mode is used, a distribution key must be specified for the user table. When a record is inserted, the system hashes it based on the distribution key and then stores it on the corresponding DN.

Select a hash distribution key based on the following principles:

1. **The values of the distribution key should be discrete so that data can be evenly distributed on each DN.** You can select the primary key of the table as the distribution key. For example, for a person information table, choose the ID number column as the distribution key.
2. **Do not select the column that has a constant filter.** For example, if a constant constraint (for example, zqdh= '000001') exists on the **zqdh** column in some queries on the **dwcjk** table, you are not advised to use **zqdh** as the distribution key.
3. **With the above principles met, you can select join conditions as distribution keys**, so that join tasks can be pushed down to DNs for execution, reducing the amount of data transferred between the DNs.

   For a hash table, an inappropriate distribution key may cause data skew or poor I/O performance on certain DNs. Therefore, you need to check the table to ensure that data is evenly distributed on each DN. You can run the following SQL statements to check for data skew:

   ```
   select
   xc_node_id, count(1)
   from tablename
   group by xc_node_id
   order by xc_node_id desc;
   ```

   xc_node_id corresponds to a DN. Generally, **over 5% difference between the amount of data on different DNs is regarded as data skew. If the difference is over 10%, choose another distribution key.**

4. You are not advised to add a column as a distribution key, especially add a new column and use the SEQUENCE value to fill the column. (Sequences may cause performance bottlenecks and unnecessary maintenance costs.)

View the data in the table.

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

Deleting Data in a Table

CAUTION

Exercise caution when running the **DROP TABLE** and **TRUNCATE TABLE** statements. After a table is deleted, data cannot be restored.

- Delete the **customer_t1** table from the database.
```
DROP TABLE customer_t1;
```
- You can use **DELETE** or **TRUNCATE** to clear rows in a table without removing the definition of the table.

  Delete all rows from the **customer_t1** table.
```
TRUNCATE TABLE customer_t1;
```
  Delete all rows from the **customer_t1** table.
```
DELETE FROM customer_t1;
```
  Delete all records whose **c_customer_sk** is **3869** from the **customer_t1** table.
```
DELETE FROM customer_t1 WHERE c_customer_sk = 3869;
```

3.5 Creating and Managing GaussDB(DWS) 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 partition cable, and a physical piece is called a partition. Data is stored on these smaller physical pieces, namely, partitions, instead of the larger logical partitioned table. During conditional query, the system scans only the partitions that meet the conditions rather than scanning the entire table improving query performance.

Advantages of partitioned tables:

- Improved query performance. You can search in specific partitions, improving the search efficiency.
- Enhanced availability. If a partition is faulty, data in other partitions is still available.
- Improved maintainability. For expired historical data that needs to be periodically deleted, you can quickly delete it by dropping or truncate partitions.

Supported Table Partition Types

- Range partitioning: partitions are created based on a numeric range, for example, by date or price range.
- List partitioning: partitions are created based on a list of values, such as sales scope or product attribute. Only clusters of 8.1.3 and later versions support this function.

Choosing to Partition a Table

You can choose to partition a table when the table has the following characteristics:

- There are obvious ranges among the fields of the table.

  A table is partitioned based on obvious rangeable fields. Generally, columns such as date, area, and value are used for partitioning. The time column is most commonly used.
- Queries to the table have obvious range characteristics.

  If the queried data fall into specific ranges, its better tables are partitioned so that through partition pruning, only the queried partition needs to be scanned, improving data scanning efficiency and reducing the I/O overhead of data scanning.
- The table contains a large amount of data.

  Scanning small tables does not take much time, therefore the performance benefits of partitioning are not significant. Therefore, you are advised to partition only large tables. In column-store table, each column is an independent file storage unit, and the minimum storage unit CU can store 60,000 rows of data. Therefore, for column-store partitioned tables, it is recommended that the data volume in each partition be greater than or equal to the number of DNs multiplied by 60,000.

Creating a Range Partitioned Table

Example: Create a table **web_returns_p1** partitioned by the range **wr_returned_date_sk**.

```
CREATE TABLE web_returns_p1
(
    wr_returned_date_sk       integer,
    wr_returned_time_sk       integer,
    wr_item_sk                  integer NOT NULL,
    wr_refunded_customer_sk   integer
)
WITH (orientation = column)
DISTRIBUTE BY HASH (wr_item_sk)
PARTITION BY RANGE (wr_returned_date_sk)
(
    PARTITION p2016 VALUES LESS THAN(20161231),
    PARTITION p2017 VALUES LESS THAN(20171231),
    PARTITION p2018 VALUES LESS THAN(20181231),
    PARTITION p2019 VALUES LESS THAN(20191231),
    PARTITION pxxxx VALUES LESS THAN(maxvalue)
);
```

Create partitions in batches, with fixed partition ranges. The following example can be used:

```
CREATE TABLE web_returns_p2
(
    wr_returned_date_sk       integer,
    wr_returned_time_sk       integer,
    wr_item_sk                  integer NOT NULL,
    wr_refunded_customer_sk   integer
)
WITH (orientation = column)
DISTRIBUTE BY HASH (wr_item_sk)
PARTITION BY RANGE(wr_returned_date_sk)
(
    PARTITION p2016 START(20161231) END(20191231) EVERY(10000),
    PARTITION p0 END(maxvalue)
);
```

Partition the table **web_returns_p2** by date and time, using time as the partition key.

```
CREATE TABLE web_returns_p2
(
   id integer,
   idle numeric,
   IO numeric,
   scope text,
   IP text,
   time timestamp
)
 WITH (TTL='7 days',PERIOD='1 day')
PARTITION BY RANGE(time)
 (
   PARTITION P1 VALUES LESS THAN('2022-01-05 16:32:45'),
   PARTITION P2 VALUES LESS THAN('2022-01-06 16:56:12')
 );
```

Creating a List Partitioned Table

A list partitioned table can use any column that allows value comparison as the partition key column. When creating a list partitioned table, you must declare the value partition for each partition.

Example: Create a list partitioned table **sales_info**.

```
CREATE TABLE sales_info
(
sale_time  timestamptz,
period     int,
city       text,
price      numeric(10,2),
remark     varchar2(100)
)
DISTRIBUTE BY HASH(sale_time)
PARTITION BY LIST (period, city)
```

```
(
PARTITION province1_202201 VALUES (('202201', 'city1'), ('202201', 'city2')),
PARTITION province2_202201 VALUES (('202201', 'city3'), ('202201', 'city4'), ('202201', 'city5')),
PARTITION rest VALUES (DEFAULT)
);
```

Partitioning an Existing Table

A table can be partitioned only when it is created. If you want to partition a table, you must create a partitioned table, load the data in the original table to the partitioned table, delete the original table, and rename the partitioned table as the name of the original table. You must also re-grant permissions on the table to users. For example:

```
CREATE TABLE web_returns_p2
(
    wr_returned_date_sk       integer,
    wr_returned_time_sk       integer,
    wr_item_sk                integer NOT NULL,
    wr_refunded_customer_sk   integer
)
WITH (orientation = column)
DISTRIBUTE BY HASH (wr_item_sk)
PARTITION BY RANGE(wr_returned_date_sk)
(
    PARTITION p2016 START(20161231) END(20191231) EVERY(10000),
    PARTITION p0 END(maxvalue)
);
INSERT INTO web_returns_p2 SELECT * FROM web_returns_p1;
DROP TABLE web_returns_p1;
ALTER TABLE web_returns_p2 RENAME TO web_returns_p1;
GRANT ALL PRIVILEGES ON web_returns_p1 TO dbadmin;
GRANT SELECT ON web_returns_p1 TO jack;
```

Adding a Partition

Run the **ALTER TABLE** statement to add a partition to a partitioned table. For example, to add partition **P2020** to the **web_returns_p1** table, run the following command:

```
ALTER TABLE web_returns_p1 ADD PARTITION P2020 VALUES LESS THAN (20201231);
```

Splitting a Partition

The syntax for splitting a partition varies between a range partitioned table and a list partitioned table.

- Run the **ALTER TABLE** statement to split a partition in a range partitioned table. For example, the partition **pxxxx** of the table **web_returns_p1** is split into two partitions **p2020** and **p20xx** at the splitting point **20201231**.
  ```
  ALTER TABLE web_returns_p1 SPLIT PARTITION pxxxx AT(20201231) INTO (PARTITION
  p2020,PARTITION p20xx);
  ```
- Run the **ALTER TABLE** statement to split a partition in a list partitioned table. For example, split the partition **province2_202201** of table **sales_inf** into two partitions **province3_202201** and **province4_202201**.
  ```
  ALTER TABLE sales_info SPLIT PARTITION province2_202201 VALUES(('202201', 'city5')) INTO
  (PARTITION province3_202201,PARTITION province4_202201);
  ```

Merging Partitions

Run the **ALTER TABLE** statement to merge two partitions in a partitioned table. For example, merge partitions **p2016** and **p2017** of table **web_returns_p1** into one partition **p20162017**.

```
ALTER TABLE web_returns_p1 MERGE PARTITIONS p2016,p2017 INTO PARTITION p20162017;
```

Deleting a Partition

Run the **ALTER TABLE** statement to delete a partition from a partitioned table. For example, run the following command to delete partition **P2020** from the **web_returns_p1** table:

```
ALTER TABLE web_returns_p1 DROP PARTITION P2020;
```

Querying a Partition

- Query partition **p2019**.
  ```
  SELECT * FROM web_returns_p1 PARTITION (p2019);
  SELECT * FROM web_returns_p1 PARTITION FOR (20201231);
  ```
- View partitioned tables using the system catalog **dba_tab_partitions**.
  ```
  SELECT * FROM dba_tab_partitions where table_name='web_returns_p1';
  ```

Deleting a Partitioned Table

Run the **DROP TABLE** statement to delete a partitioned table.

```
DROP TABLE web_returns_p1;
```

3.6 Creating and Managing GaussDB(DWS) Indexes

Indexes accelerate the data access speed but also add the processing time of the insert, update, and delete operations. Therefore, before creating an index, consider whether it is necessary and determine the columns where indexes will be created. You can determine whether to add 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 sorted.

Index type

- **btree**: The B-tree index uses a structure that is similar to the B+ tree structure to store data key values, facilitating index search. **btree** supports 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.
- **Psort**: psort index. It is used to perform partial sort on column-store tables.

Row-based tables support the following index types: **btree** (default), **gin**, and **gist**. Column-based tables support the following index types: **Psort** (default), **btree**, and **gin**.

NOTE

Create a B-tree index for point queries.

Index Selection Principles

Indexes are created based on columns in database tables. When creating indexes, you need to determine the columns, which can be:

- Columns that are frequently searched: The search efficiency can be improved.
- The uniqueness of the columns and the data sequence structures is ensured.
- Columns that usually function as foreign keys and are used for connections. Then the connection efficiency is improved.
- Columns that are usually searched for by a specified scope. These indexes have already been arranged in a sequence, and the specified scope is contiguous.
- Columns that need to be arranged in a sequence. These indexes have already been arranged in a sequence, so the sequence query time is accelerated.
- Columns that usually use the WHERE clause. Then the condition decision efficiency is increased.
- Fields that are frequently used after keywords, such as **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. Therefore, data operations increase. Therefore, delete unnecessary indexes periodically.

Creating an Index

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

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. Therefore, data operations increase. Therefore, delete unnecessary indexes periodically.

Table 3-3 Indexing Method

Indexing Method	Description
Unique index	Refers to an index that constrains the uniqueness of an index attribute or an attribute group. If a table declares unique constraints or primary keys, GaussDB(DWS) automatically creates unique indexes (or composite indexes) for columns that form the primary keys or unique constraints. Currently, only B-tree can create a unique index in GaussDB(DWS).
Composite index	Refers to an index that can be defined for multiple attributes of a table. Currently, composite indexes can be created only for B-tree in GaussDB(DWS) and a maximum of 32 columns can share a composite index.
Partial index	Refers to an index that can be created for subsets of a table. This indexing method contains only tuples that meet condition expressions.
Expression index	Refers to an index that is built on a function or an 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.

- Run the following command to create an ordinary table:
  ```
  CREATE TABLE tpcds.customer_address_bak AS TABLE tpcds.customer_address;
  ```
- Create a common index.

  You need to query the following information in the **tpcds.customer_address_bak** table:
  ```
  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 low. 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 delete operation performance in the database.

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

  If a table declares a unique constraint or primary key, GaussDB(DWS) automatically creates a unique index (possibly a multi-column index) on the columns that form the primary key or unique constraint. If no unique constraint or primary key is specified during table creation, you can run the CREATE INDEX statement to create an index.
  ```
  CREATE UNIQUE INDEX unique_index ON tpcds.customer_address_bak(ca_address_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 command:
  ```
  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 multiple-column index on **ca_address_sk** and **ca_street_number** columns:
  ```
  CREATE INDEX more_column_index ON
  tpcds.customer_address_bak(ca_address_sk ,ca_street_number );
  ```
- Create a partition index.

  If you only want to find records whose **ca_address_sk** is **5050**, you can create a partial index to facilitate your query.
  ```
  CREATE INDEX part_index ON tpcds.customer_address_bak(ca_address_sk) WHERE ca_address_sk =
  5050;
  ```
- Create an expression index.

  Assume you need to frequently query records with **ca_street_number** smaller than **1000**, run the following command:
  ```
  SELECT * FROM tpcds.customer_address_bak WHERE trunc(ca_street_number) < 1000;
  ```
  The following expression index can be created for this query task:
  ```
  CREATE INDEX para_index ON tpcds.customer_address_bak (trunc(ca_street_number));
  ```

Querying an Index

- Run the following command to query all indexes defined by the system and users:
  ```
  SELECT RELNAME FROM PG_CLASS WHERE RELKIND='i';
  ```
- Run the following command to query information about a specified index:
  ```
  \di+ index_wr_returned_date_sk
  ```

Recreating an Index

- Recreate the index **index_wr_returned_date_sk**.
  ```
  REINDEX INDEX index_wr_returned_date_sk;
  ```
- Recreate all indexes of a table.
  ```
  REINDEX TABLE tpcds.customer_address_bak;
  ```

Deleting an Index

You can use the **DROP INDEX** statement to delete indexes.
```
DROP INDEX index_wr_returned_date_sk;
```

3.7 Creating and Using GaussDB(DWS) Sequences

A sequence is a database object that generates unique integers according to a certain rule and is usually used to generate primary key values.

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

- Set the data type of a column to sequence integer. A sequence will be automatically created by the database for this column.
- Use **CREATE SEQUENCE** to create a new sequenc. Use the **nextval('***sequence_name***')** function to increment the sequence and return a

new value. Specify the default value of the column as the sequence value returned by the **nextval('**sequence_name**')** function. In this way, this column can be used as a unique identifier.

Creating a Sequence.

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

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

Method 2: Create a sequence and set the initial value of the **nextval**('*sequence_name*') function to the default value of a column. You can cache a specific number of sequence values to reduce the requests to the GTM, improving the performance.

1. Create a sequence.
   ```
   CREATE SEQUENCE seq1 cache 100;
   ```
2. Set the initial value of the **nextval**('*sequence_name*') function to the default value of a column.
   ```
   CREATE TABLE T2
   (
       id   int not null default nextval('seq1'),
       name text
   );
   ```

NOTE

Methods 1 and 2 are similar except that method 2 specifies cache for the sequence. A sequence using cache has holes (non-consecutive values, for example, 1, 4, 5) and cannot keep the order of the values. After a sequence is deleted, its sub-sequences will be deleted automatically. A sequence shared by multiple columns is not forbidden in a database, but you are not advised to do that.

Currently, the preceding two methods cannot be used for existing tables.

Modifying a Sequence

The **ALTER SEQUENCE** statement changes the attributes of an existing sequence, including the owner, owning column, and maximum value.

- Associate the sequence with a column.

  The sequence will be deleted when you delete the column or the table where the column resides.
  ```
  ALTER SEQUENCE seq1 OWNED BY T2.id;
  ```
- Modify the maximum value of **serial** to **300**.
  ```
  ALTER SEQUENCE seq1 MAXVALUE 300;
  ```

Deleting a Sequence

Run the **DROP SEQUENCE** command to delete a sequence. For example, to delete the sequence named **seq1**, run the following command:

```
DROP SEQUENCE seq1;
```

Precautions

Sequence values are generated by the GTM. By default, each request for a sequence value is sent to the GTM. The GTM calculates the result of the current value plus the step and then returns the result. As GTM is a globally unique node, generating default sequence numbers can cause performance issues. For operations that need frequent sequence number generation, such as bulkload data import, this is not recommended. For example, the **INSERT FROM SELECT** statement has poor performance in the following scenario:

```
CREATE SEQUENCE newSeq1;
CREATE TABLE newT1
        (
          id   int not null default nextval('newSeq1'),
          name text
         );
INSERT INTO newT1(name) SELECT name from T1;
```

To improve the performance, run the following statements (assume that data of 10,000 rows will be imported from *T1* to *newT1*):

```
INSERT INTO newT1(id, name) SELECT id,name from T1;
SELECT SETVAL('newSeq1',10000);
```

NOTE

Rollback is not supported by sequence functions, including **nextval()** and **setval()**. The value of the setval function immediately takes effect on nextval in the current session in any cases and take effect in other sessions only when no cache is specified for them. If cache is specified for a session, it takes effect only after all the cached values have been used. To avoid duplicate values, use setval only when necessary. Do not set it to an existing sequence value or a cached sequence value.

If BulkLoad is used, set sufficient cache for *newSeq1* and do not set **Maxvalue** or **Minvalue**. To improve the performance, database may push down the invocation of **nextval**('*sequence_name*') to DNs. Currently, the concurrent connection requests that can be processed by the GTM are limited. If there are too many DNs, a large number of concurrent connection requests will be sent to the GTM. In this case, you need to limit the concurrent connection of BulkLoad to save the GTM connection resources. If the target table is a replication table (**DISTRIBUTE BY REPLICATION**), pushdown cannot be performed. If the data volume is large, this will be a disaster for the database. In addition, the database space may be exhausted. After the import is complete, do **VACUUM FULL**. Therefore, you are not advised to use sequences when BulkLoad is used.

After a sequence is created, a single-row table is maintained on each node to store the sequence definition and value, which is obtained from the last interaction with the GTM rather than updated in real time. The single-row table on a node does not update when other nodes request a new value from the GTM or when the sequence is modified using **setval**.

3.8 Creating and Managing GaussDB(DWS) Views

Views allow users to save queries. Views are not physically stored on disks. Queries to a view run as subqueries. A database only stores the definition of a view and does not store its data. The data is still stored in the original 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.

Creating a view

Run the **CREATE VIEW** command to create a view.

```
CREATE OR REPLACE VIEW MyView AS SELECT * FROM tpcds.customer WHERE c_customer_sk < 150;
```

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.

View Details

- View the *MyView* view. Real-time data will be returned.
  ```
  SELECT * FROM myview;
  ```
- Run the following command to query the views in the current user:
  ```
  SELECT * FROM user_views;
  ```
- Run the following command to query all views:
  ```
  SELECT * FROM dba_views;
  ```
- View details about a specified view.

  Run the following command to view details about the dba_users view:
  ```
  \d+ dba_users
                      View "PG_CATALOG.DBA_USERS"
    Column  |         Type          | Modifiers | Storage  | Description
  ----------+-----------------------+-----------+----------+-------------
   USERNAME | CHARACTER VARYING(64) |           | extended |
  View definition:
   SELECT PG_AUTHID.ROLNAME::CHARACTER VARYING(64) AS USERNAME
     FROM PG_AUTHID;
  ```

Rebuilding a View

Run the **ALTER VIEW** command to rebuild a view without entering query statements.

```
ALTER VIEW myview REBUILD;
```

Deleting a View

Run the **DROP VIEW** command to delete a view.

```
DROP VIEW myview;
```

DROP VIEW ... The **CASCADE** command can be used to delete objects that depend on the view. For example, view A depends on view B. If view B is deleted, view A will also be deleted. Without the CASCADE option, the **DROP VIEW** command will fail.

3.9 Creating and Managing GaussDB(DWS) Scheduled Tasks

GaussDB(DWS) allows users to create scheduled tasks, which are automatically executed at specified time points, reducing O&M workload.

Database complies with the Oracle scheduled task function using the DBMS.JOB interface, which can be used to create scheduled tasks, execute tasks automatically, delete a task, and modify task attributes(including task ID, enable/disable a task, the task triggering time/interval and task contents).

NOTE

- The hybrid data warehouse (standalone) does not support scheduled tasks.
- The execution statements of scheduled tasks are not recorded in the **Real-time Top SQL** logs. The statements can be recorded only in versions later than 8.2.1.
- By default, GaussDB(DWS) uses the UTC time. The execution time of the scheduled task needs to be converted to the time zone of the user.

Periodic Task Management

Step 1 Creates a test table.

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

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

```
CREATE TABLE
```

Step 2 Create the customized storage procedure.

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

If the following information is displayed, the procedure has been created.

```
CREATE PROCEDURE
```

Step 3 Create a task.

- Create a task with unspecified **job_id** and execute the **PRC_JOB_1** storage procedure every two minutes.

```
call dbms_job.submit('call public.prc_job_1(); ', sysdate, 'interval ''1 minute''', :a);
job
-----
1
(1 row)
```

- Create task with specified job_id.

```
call dbms_job.isubmit(2,'call public.prc_job_1(); ', sysdate, 'interval ''1 minute''');
isubmit
---------

(1 row)
```

Step 4 View the created task information about the current user in the **USER_JOBS** view.

Only the system administrator can access this system view. For details about the fields, see **Table 14-295**.

```
select job,dbname,start_date,last_date,this_date,next_date,broken,status,interval,failures,what from
user_jobs;
 job |  dbname  |        start_date        |        last_date        |        this_date        |      next_date      |
broken | status |      interval       | failures |          what
```

```
-----+----------+----------------------------+----------------------------+----------------------------
+---------------------+--------+--------+---------------------+----------+----------------
-----------
   1 | db_demo | 2022-03-25 07:58:01.829436 | 2022-03-25 07:58:03.174817 | 2022-03-25 07:58:01.829436 |
2022-03-25 07:59:01 | n      | s      | interval '1 minute' |        0 | call public.prc
_job_1();
   2 | db_demo | 2022-03-25 07:58:15.893383 | 2022-03-25 07:58:16.608959 | 2022-03-25 07:58:15.893383 |
2022-03-25 07:59:15 | n      | s      | interval '1 minute' |        0 | call public.prc
_job_1();
(2 rows)
```

Step 5 Stop a task.

```
call dbms_job.broken(1,true);
broken
--------

(1 row)
```

Step 6 Start a task.

```
call dbms_job.broken(1,false);
broken
--------

(1 row)
```

Step 7 Modify attributes of a task.

- Modify the **Next_date** parameter information about a task. For example, change the value of **Next_date** of Job1 to 1 hour.
```
call dbms_job.next_date(1, sysdate+1.0/24);
next_date
-----------

(1 row)
```
- Modify the **Interval** parameter information of a task. For example, change the value of **Interval** of Job1 to 1 hour.
```
call dbms_job.interval(1,'sysdate + 1.0/24');
interval
----------

(1 row)
```
- Modify the **What** parameter information of a **JOB**. For example, change **What** of **Job1** to **insert into public.test values(333, sysdate+5)**.
```
call dbms_job.what(1,'insert into public.test values(333, sysdate+5);');
what
------

(1 row)
```
- Modify **Next_date**, **Interval**, and **What** parameter information of **JOB**.
```
call dbms_job.change(1, 'call public.prc_job_1();', sysdate, 'interval ''1 minute''');
change
--------

(1 row)
```

Step 8 Delete a job.

```
call dbms_job.remove(1);
remove
--------

(1 row)
```

Step 9 Set job permissions.

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

----End

3.10 Viewing GaussDB(DWS) System Catalogs

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

Querying Database Tables

For example, 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 | nodegroup | tempspacelimit | spillspacelim
it
---------+----------+-------------+----------+-----------+---------+----------+----------+----------+--------------
+--------+------------+-----------+-----------+----------------+--------------
---
```

```
 Ruby     |       10 | t          | t        | t        | t       | ******** |          |          | default_pool |      0 |
|          |          |              |
 dbadmin |    16393 | f          | f        | f        | f       | ******** |          |          | default_pool |      0 |
|          |          |              |
 lily    |    16691 | f          | f        | f        | f       | ******** |          |          | default_pool |      0 |
|          |          |              |
 jack    |    70694 | f          | f        | f        | f       | ******** |          |          | default_pool |      0 |
|          |          |              |
(4 rows)
```

GaussDB(DWS) uses Ruby to perform routine management and maintenance. You can add **WHERE usesysid > 10** to the **SELECT** statement to filter queries so that only specified user names are displayed.

```
SELECT * FROM pg_user WHERE usesysid > 10;
 usename | usesysid | usecreatedb | usesuper | usecatupd | userepl |  passwd  | valbegin | valuntil |
respool    | parent | spacelimit | useconfig | nodegroup | tempspacelimit | spillspacelim
it
---------+----------+-------------+----------+-----------+---------+----------+----------+----------+--------------
+--------+------------+-----------+-----------+----------------+--------------
---
 dbadmin |    16393 | f          | f        | f        | f       | ******** |          |          | default_pool |      0 |
|          |          |              |
 lily    |    16691 | f          | f        | f        | f       | ******** |          |          | default_pool |      0 |
|          |          |              |
 jack    |    70694 | f          | f        | f        | f       | ******** |          |          | default_pool |      0 |
|          |          |              |
(3 rows)
```

Viewing and Stopping the Running Query Statements

You can view the running query statements in the **PG_STAT_ACTIVITY** view. Do as follows:

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, query statuses, and PIDs of the running query statements:

```
SELECT datname, usename, state,pid FROM pg_stat_activity;
```

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 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." is returned. Attempting reset: Succeeded."

```
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

4 Syntax Compatibility Differences Among Oracle, Teradata, and MySQL

In GaussDB(DWS), **DBCOMPATIBILITY** can be set to **TD**, **Oracle**, or **MySQL** to be compatible with Teradata, Oracle, or MySQL syntax, respectively. Syntax behavior varies with the three modes.

The database compatibility model can be specified using the **DBCOMPATIBILITY** parameter when creating a database. For details, refer to the **CREATE DATABASE** syntax.

```
CREATE DATABASE ora_compatible_db DBCOMPATIBILITY 'ORA';
```

Table 4-1 Compatibility differences

Compatibility Item	Oracle	Teradata	MySQL
Empty string	Only **null** is available.	An empty string is distinguished from **null**.	An empty string is distinguished from **null**.
Conversion of an empty string to a number	Null	0	0
Automatic truncation of overlong characters	Not supported	Supported (set GUC parameter **td_compatible_truncation** to **ON**)	Not supported

Compatibility Item	Oracle	Teradata	MySQL
null concatenation	Returns a non-null object after combining a non-null object with **null**. For example, **'abc'||null** returns **'abc'**.	The **strict_text_concat_td** option is added to the GUC parameter **behavior_compat_options** to be compatible with the Teradata behavior. After the null type is concatenated, **null** is returned. For example, **'abc'|| null** returns **null**.	Compatible with MySQL behavior. After the null type is concatenated, **null** is returned. For example, **'abc'|| null** returns **null**.
Concatenation of the char(n) type	Removes spaces and placeholders on the right when the char(n) type is concatenated. For example, **cast('a' as char(3))||'b'** returns **'ab'**.	After the **bpchar_text_without_rtrim** option is added to the GUC parameter **behavior_compat_options**, when the char(n) type is concatenated, spaces are reserved and supplemented to the specified length *n*. Currently, ignoring spaces at the end of a string for comparison is not supported. If the concatenated string contains spaces at the end, the comparison is space-sensitive. For example, **cast('a' as char(3))||'b'** returns **'a b'**.	Removes spaces and placeholders on the right.
concat(str1,str2)	Returns the concatenation of all non-null strings.	Returns the concatenation of all non-null strings.	If an input parameter is **null**, **null** is returned.

Compatibility Item	Oracle	Teradata	MySQL
left and right processing of negative values	Returns all characters except the first and last |n| characters.	Returns all characters except the first and last |n| characters.	Returns an empty string.
lpad(string text, length int [, fill text]) rpad(string text, length int [, fill text])	Fills up the string to the specified **length** by appending the **fill** characters (a space by default). If the string is already longer than **length** then it is truncated (on the right). If **fill** is an empty string or **length** is a negative number, **null** is returned.	If **fill** is an empty string and the string length is less than the specified **length**, the original string is returned. If **length** is a negative number, an empty string is returned.	If **fill** is an empty string and the string length is less than the specified **length**, an empty string is returned. If **length** is a negative number, **null** is returned.
substr(str, s[, n])	If **s** is set to 0, the first n characters are returned.	If **s** is set to 0, the first n characters are returned.	If **s** is set to 0, an empty string is returned.

substring(str, s[, n])
substring(str [from s] [for n])
	If **s** is set to 0, the first n - 1 characters are returned.
	If **s** is < 0, the first s + n - 1 characters are returned.
	If **n** is < 0, an error is reported.
		If **s** is set to 0, the first n - 1 characters are returned.
		If **s** is < 0, the first s + n - 1 characters are returned.
		If **n** is < 0, an error is reported.
			If **s** is set to 0, an empty string is returned.
			If **s** is < 0, n characters starting from the last |s| character are truncated.
			If **n** is < 0, an empty string is returned.

trim, ltrim, rtrim, btrim(string[, characters])	Removes the longest string that contains only the characters (a space by default) in the *characters* from a specified position of the *string*.	Removes the longest string that contains only the characters (a space by default) in the *characters* from a specified position of the *string*.	Removes the string that is equivalent to characters (a space by default) from a specified position of the *string*.

Compatibility Item	Oracle	Teradata	MySQL
log(x)	Returns the logarithm with 10 as the base.	Returns the logarithm with 10 as the base.	Returns the natural logarithm.
mod(x, 0)	Returns x if the divisor is 0.	Returns x if the divisor is 0.	Reports an error if the divisor is 0.
date data type	Converts the date data type to the timestamp data type which stores year, month, day, hour, minute, and second values.	Stores year and month values.	Stores year and month values.
to_char(date)	The maximum value of the input parameter can only be the maximum value of the timestamp type. The maximum value of the date type is not supported. The return value is of the timestamp type.	The maximum value of the input parameter can only be the maximum value of the timestamp type. The maximum value of the date type is not supported. The return value is of the date type in YYYY/MM/DD format. (The GUC parameter **convert_empty_str_to_null_td** is enabled.)	Only the timestamp type and the date type support the maximum input value. The return value is of the date type.
to_date, to_timestamp, and to_number processing of empty strings	Returns **null**.	Returns **null**. (The **convert_empty_str_to_null_td** parameter is enabled.)	**to_date** and **to_timestamp** returns **null**. If the parameter passed to **to_number** is an empty string, **0** is returned.
Return value types of last_day and next_day	Returns values of the timestamp type.	Returns values of the timestamp type.	Returns values of the date type.

Compatibility Item	Oracle	Teradata	MySQL
Return value type of add_months	Returns values of the timestamp type.	Returns values of the timestamp type.	If the input parameter is of the date type, the return value is of the date type. If the input parameter is of the timestamp type, the return value is of the timestamp type. If the input parameter is of the timestamptz type, the return value is of the timestamptz type.
CURRENT_TIME CURRENT_TIME(p)	Obtains the time of the current transaction. The return value is of the timetz type.	Obtains the time of the current transaction. The return value is of the timetz type.	Obtains the execution time of the current statement. The return value is of the time type.
CURRENT_TIMESTAMP CURRENT_TIMESTAMP(p)	Obtains the execution time of the current statement. The return value is of the timestamptz type.	Obtains the execution time of the current statement. The return value is of the timestamptz type.	Obtains the execution time of the current statement. The return value is of the timestamp type.
LOCALTIME LOCALTIME(p)	Obtains the time of the current transaction. The return value is of the time type.	Obtains the time of the current transaction. The return value is of the time type.	Obtains the execution time of the current statement. The return value is of the timestamp type.
LOCALTIMESTAMP LOCALTIMESTAMP(p)	Obtains the time of the current transaction. The return value is of the timestamp type.	Obtains the time of the current transaction. The return value is of the timestamp type.	Obtains the execution time of the current statement. The return value is of the timestamp type.

Compatibility Item	Oracle	Teradata	MySQL
SYSDATE SYSDATE(p)	Obtains the execution time of the current statement. The return value is of the timestamp(0) type.	Obtains the execution time of the current statement. The return value is of the timestamp(0) type.	Obtains the current system time. The return value is of the timestamp(0) type. This function cannot be pushed down. You are advised to use current_date instead.
now()	Obtains the time of the current transaction. The return value is of the timestamptz type.	Obtains the time of the current transaction. The return value is of the timestamptz type.	Obtains the statement execution time. The return value is of the timestamptz type.
Operator ^	Performs exponentiation.	Performs exponentiation.	Performs the exclusive OR operation.
Expressions GREATEST and LEAST	Returns the comparison results of all non-null input parameters.	Returns the comparison results of all non-null input parameters.	If an input parameter is **null**, **null** is returned.
Different input parameter types of CASE, COALESCE, IF, and IFNULL expressions	Reports error.	Is compatible with behavior of Teradata and supports type conversion between digits and strings. For example, if input parameters for COALESCE are of INT and VARCHAR types, the parameters are resolved as VARCHAR type.	Is compatible with behavior of MySQL and supports type conversion between strings and other types. For example, if input parameters for COALESCE are of DATE, INT, and VARCHAR types, the parameters are resolved as VARCHAR type.

5 GaussDB(DWS) Database Security Management

5.1 GaussDB(DWS) User and Permissions Management

5.1.1 GaussDB(DWS) Database User Types

Without separation of permissions, GaussDB(DWS) supports two types of database accounts: administrator and common user. For details about user types and permissions under separation of permissions, see **Separation of Duties in GaussDB(DWS)**.

- The administrator can manage all common users and databases.
- Common users can connect to and access the database, and perform specific database operations and execute SQL statements after being authorized.

Users are authenticated when they log in to the GaussDB(DWS) database. 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 to these databases.

Database User Types

Table 5-1 Database user types

User Type	Description	Allowed Operations	How to Create

Administrator **dbadmin**	An administrator, also called a system administrator, is an account with the **SYSADMIN** attribute.	If separation of permissions is not enabled, this account has the highest permission in the system and can perform all operations. The system administrator has the same permissions as the object owner.	● User **dbadmin** created during cluster creation on the GaussDB(DWS) management console is a system administrator.
			● Use the **CREATE USER** or **ALTER USER** syntax to create an administrator.
			`CREATE USER` *sysadmin* `WITH SYSADMIN password '`*{Password}*`';`
			`ALTER USER` *u1* `SYSADMIN;`

Common user	Common user	● Use a tool to connect to the database. ● Have the attributes of specific database system operations, such as **CREATEDB**, **CREATEROLE**, and **SYSADMIN**. ● Access database objects. ● Run SQL statements.	Run the **CREATE USER** syntax to create a common user. `CREATE USER` *u1* `PASSWORD '`*{Password}*`';`
	Private user	A user created with the **INDEPENDENT** attribute in non-separation-of-permissions mode. Database administrators can manage (**DROP**, **ALTER**, and **TRUNCATE**) objects of private users but cannot access (**INSERT**, **DELETE**, **SELECT**, **UPDATE**, **COPY**, **GRANT**, **REVOKE**, and **ALTER OWNER**) the objects before being authorized.	Use the **CREATE USER** syntax to create a private user. `CREATE USER` *user_independent* `WITH INDEPENDENT IDENTIFIED BY '`*{Password}*`';`

5.1.2 GaussDB(DWS) Database User Management

You can use **CREATE USER** and **ALTER USER** to create and manage database users.

- In the non-**separation-of-permission** mode, a GaussDB(DWS) user account can be created and deleted only by a system administrator or a security administrator with the **CREATEROLE** attribute.
- In separation-of-permission mode, a user account can be created only by a security administrator.

Creating a User

The **CREATE USER** statement is used to create a GaussDB (DWS) user. After creating a user, you can use the user to connect to the database.

- Create common user **u1** and assign the **CREATEDB** attribute to the user.
  ```
  CREATE USER u1 WITH CREATEDB PASSWORD '{Password}';
  ```
- To create the system administrator **mydbadmin**, you need to specify the **SYSADMIN** parameter.
  ```
  CREATE USER mydbadmin sysadmin PASSWORD '{Password}';
  ```
- View the created user in the **PG_USER** view.
  ```
  SELECT * FROM pg_user;
  ```
- To view user attributes, query the system catalog **PG_AUTHID**.
  ```
  SELECT * FROM pg_authid;
  ```

Altering User Attributes

The **ALTER USER** statement is used to alter user attributes, such as changing user passwords or permissions.

Example:

- Rename user **u1** to **u2**.
  ```
  ALTER USER u1 RENAME TO u2;
  ```
- Grant the **CREATEROLE** permission to user **u1**:
  ```
  ALTER USER u1 CREATEROLE;
  ```
- For details about how to change the user password, see **Setting and Changing a Password**.

Locking a User

The **ACCOUNT LOCK** | **ACCOUNT UNLOCK** parameter in the statement is used to lock or unlock a user. A locked user cannot log in to the system. If an account is stolen or illegally accessed, the administrator can manually lock the account. After the account is secured, the administrator can manually unlock the account.

Example:

- To lock user **u1**, run the following command:
  ```
  ALTER USER u1 ACCOUNT LOCK;
  ```
- To unlock user **u1**, run the following command:
  ```
  ALTER USER u1 ACCOUNT UNLOCK;
  ```

Deleting a User

The **DROP USER** statement is used to delete one or more GaussDB(DWS) users. An administrator can delete an account that is no longer used. Deleted users cannot be restored.

- If multiple users are deleted at the same time, separate them with commas (,).
- After a user is deleted successfully, all the permissions of the user are also deleted.
- When an account to be deleted is in the active state, it is deleted after the session is disconnected.
- When **CASCADE** is specified in the **DROP USER** statement, objects such as tables that depend on the user will be deleted. That is, the objects whose owner is the user are deleted, and the authorizations of other objects to the user are also deleted.

Example:

- -- Delete user **u1**.
  ```
  DROP USER u1;
  ```
- Delete account **u2** in a cascading manner.
  ```
  DROP USER u2 CASCADE;
  ```

5.1.3 Creating a Custom Password Policy for GaussDB(DWS)

When creating or modifying a user, you need to specify a password. GaussDB(DWS) has default password complexity requirements. You can also define database account password policies.

Default GaussDB(DWS) Password Policy

By default, GaussDB(DWS) verifies the password complexity (that is, the GUC parameter **password_policy** is set to **1** by default). The default password policy requires that the password:

- Contain 8 to 32 characters.
- Contain at least three types of the following characters: uppercase letters, lowercase letters, digits, and special characters.
- Cannot be the same as the user name or the user name in reverse order, case insensitive.
- Cannot be the current password or the current password in reverse order.

User-defined Password Policy

The password policy includes the password complexity requirements, password validity period, password reuse settings, password encryption mode, and password retry and lock policies. Different policy items are controlled by the corresponding GUC parameters. For details, see **Security and Authentication (postgresql.conf)**.

Table 5-2 User-defined password policies and corresponding GUC parameters

Password Policy	Parameter	Description	Value Range	Default Value in GaussDB(DWS)
Password complexity check	password_policy	Specifies whether to check the password complexity when a GaussDB(DWS) account is created or modified.	Integer, **0** or **1** ● **0** indicates that no password complexity policy is used. Setting this parameter to **0** leads to security risks. You are advised not to set this parameter to **0**. ● **1** indicates that the default password complexity policy is used.	1
Password complexity requirement	password_min_length	Specifies the minimum password length.	An integer ranging from 6 to 999	8
	password_max_length	Specifies the maximum password length.	An integer ranging from 6 to 999	32
	password_min_uppercase	Minimum number of uppercase letters (A-Z)	An integer ranging from 0 to 999 ● **0** means no requirements. ● **1-999** indicates the minimum number of uppercase letters in the password.	0
	password_min_lowercase	Minimum number of lowercase letters (a-z)	An integer ranging from 0 to 999 ● **0** means no requirements. ● **1-999** indicates the minimum number of lower letters in the password.	0

Password Policy	Parameter	Description	Value Range	Default Value in GaussDB(DWS)
	password_min_digital	Minimum number of digits (0-9)	An integer ranging from 0 to 999 • **0** means no requirements. • **1-999** indicates the minimum number of digits in the password.	0
	password_min_special	Minimum number of special characters (**Table 5-3** lists the special characters.)	An integer ranging from 0 to 999 • **0** means no requirements. • **1-999** indicates the minimum number of special characters in the password.	0
Password validity	password_effect_time	Password validity period When the number of days in advance a user is notified that the password is about to expire reaches the value of **password_notify_time**, the system prompts the user to change the password when the user logs in to the database.	The value is a floating point number ranging from 0 to 999. The unit is day. • **0** indicates the validity period is disabled. • A floating point number from 1 to 999 indicates the validity period of the password. When the password is about to expire or has expired, the system prompts the user to change the password.	90

Password Policy	Parameter	Description	Value Range	Default Value in GaussDB(DWS)
	password_notify_time	Specifies for how many days you are reminded of the password expiry.	The value is an integer ranging from 0 to 999. The unit is day. ● **0** indicates the reminder is disabled. ● A value ranging from 1 to 999 indicates the number of days prior to password expiration that a user will receive a notification.	7
Password reuse settings	password_reuse_time	Specifies the number of days after which the password cannot be reused.	A Floating point number ranging from 0 to 3650. The unit is day. ● **0** indicates that the password reuse days are not checked. ● A positive number indicates that the new password cannot be chosen from passwords in history that are newer than the specified number of days.	60
	password_reuse_max	Specifies the number of the most recent passwords that the new password cannot be chosen from.	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 chosen from the specified number of the most recent passwords.	0

Password Policy	Parameter	Description	Value Range	Default Value in GaussDB(DWS)
Encryption mode	password_encryption_type	Specifies the password storage encryption mode.	0, 1, 2 ● **0** indicates that passwords are encrypted in MD5 mode. The password is encrypted using MD5. This mode is not recommended for users. ● **1** indicates that passwords are encrypted with SHA-256, which is compatible with the MD5 user authentication method of the PostgreSQL client. The password is stored in ciphertext encrypted by MD5 and SHA256. ● **2** indicates that passwords are encrypted using SHA-256. The password is encrypted using SHA256.	1

Password Policy	Parameter	Description	Value Range	Default Value in GaussDB(DWS)
Retry and lock	password_lock_time	Specifies the duration for a locked account to be automatically unlocked.	A Floating point number ranging from 0 to 365. The unit is day. ● **0** indicates that the account is not automatically locked if the password verification fails. ● A positive number indicates the duration after which a locked account is automatically unlocked. **NOTE** The integral part of the value of the **password_lock_time** parameter indicates the number of days and its decimal part can be converted into hours, minutes, and seconds.	1
	failed_login_attempts	If the number of incorrect password attempts reaches the value of failed_login_attempts, the account is locked and will be automatically unlocked in X (which indicates the value of password_lock_time) seconds.	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 value of **failed_login_attempts**.	10

Table 5-3 Special characters

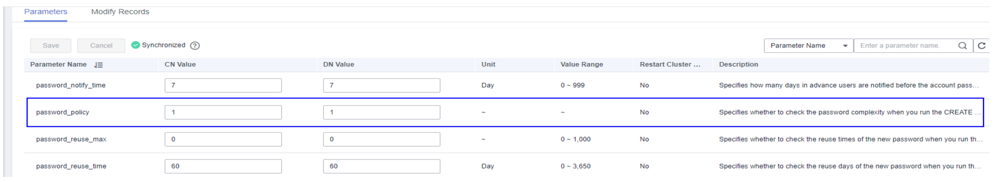
No.	Character	No.	Character	No.	Character	No.	Character
1	~	9	*	17	|	25	<
2	!	10	(	18	[	26	.
3	@	11	)	19	{	27	>
4	#	12	-	20	}	28	/
5	$	13	_	21	]	29	?
6	%	14	=	22	;	-	-
7	^	15	+	23	:	-	-
8	&	16	\	24	,	-	-

Example of User-defined Password Policies

Example 1: Configure the password complexity parameter password_policy.

1. Log in to the GaussDB(DWS) management console.
2. In the navigation pane on the left, choose **Clusters**.
3. In the cluster list, find the target cluster and click the cluster name. The **Cluster Information** page is displayed.
4. Click the **Parameters** tab, change the value of **password_policy**, and click **Save**. The **password_policy** parameter takes effect immediately after being modified. You do not need to restart the cluster.

Figure 5-1 password_policy

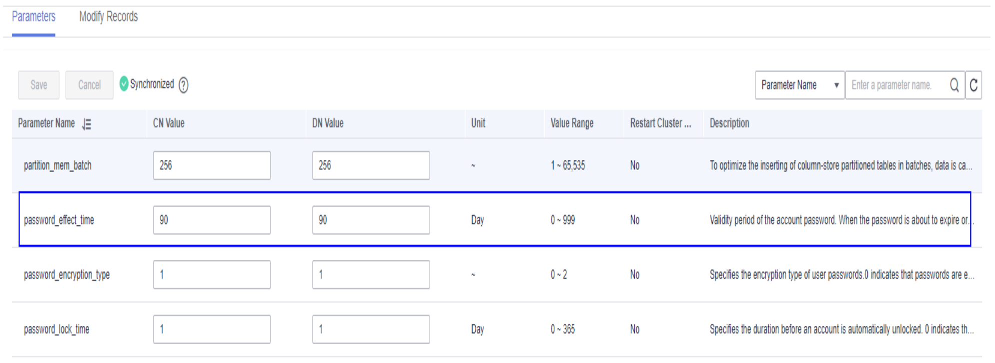


Parameter Name	CN Value	DN Value	Unit	Value Range	Restart Cluster ...	Description
password_notify_time	7	7	Day	0 ~ 999	No	Specifies how many days in advance users are notified before the account pass...
password_policy	1	1	~	~	No	Specifies whether to check the password complexity when you run the CREATE ...
password_reuse_max	0	0	~	0 ~ 1,000	No	Specifies whether to check the reuse times of the new password when you run th...
password_reuse_time	60	60	Day	0 ~ 3,650	No	Specifies whether to check the reuse days of the new password when you run th...

Example 2: Configure password_effect_time for password validity period.

1. Log in to the GaussDB(DWS) management console.
2. In the navigation pane on the left, choose **Clusters**.
3. In the cluster list, find the target cluster and click the cluster name. The **Cluster Information** page is displayed.
4. Click the **Parameters** tab, change the value of **password_effect_time**, and click **Save**. The modification of **password_effect_time** takes effect immediately. You do not need to restart the cluster.

Figure 5-2 password_effect_time



Parameter Name	CN Value	DN Value	Unit	Value Range	Restart Cluster ...	Description
partition_mem_batch	256	256	~	1 ~ 65,535	No	To optimize the inserting of column-store partitioned tables in batches, data is ca...
password_effect_time	90	90	Day	0 ~ 999	No	Validity period of the account password. When the password is about to expire or...
password_encryption_type	1	1	~	0 ~ 2	No	Specifies the encryption type of user passwords.0 indicates that passwords are e...
password_lock_time	1	1	Day	0 ~ 365	No	Specifies the duration before an account is automatically unlocked. 0 indicates th...

Setting and Changing a Password

- Both 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 the user **user1**, connect to the database as the administrator and run the following command:

  ```
  ALTER USER user1 IDENTIFIED BY 'newpassword' REPLACE 'oldpassword';
  ```

  NOTE

  The password must meet input requirements, or the execution will fail.

- An administrator can change its own password and other accounts' passwords. With the permission for changing other accounts' passwords, the administrator can resolve a login failure when a user forgets its password.

  To change the password of the user **joe**, run the following command:

  ```
  ALTER USER joe IDENTIFIED BY 'password';
  ```

NOTE

- System administrators are not allowed to change passwords for each other.
- When a system administrator changes the password of a common user, the original password is not required.
- However, when a system administrator changes its own password, the original password is required.

- 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.

  To set the password of the user **joe**, run the following command:

  ```
  SET ROLE joe PASSWORD 'password';
  ```

  If the following information is displayed, the role setting has been modified:

  ```
  SET ROLE
  ```

5.1.4 GaussDB(DWS) Database Permissions Management

Permission Overview

Permissions are used to control whether a user is allowed to access a database object (including schemas, tables, functions, and sequences) to perform operations such as adding, deleting, modifying, querying, and creating a database object.

Permission management in GaussDB(DWS) falls into three categories:

- System permissions

  System permissions are also called user attributes, including **SYSADMIN**, **CREATEDB**, **CREATEROLE**, **AUDITADMIN**, and **LOGIN**.

  They can be specified only by the **CREATE ROLE** or **ALTER ROLE** syntax. The **SYSADMIN** permission 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**.
- Object permissions

  Permissions on a database object (table, view, column, database, function, schema, or tablespace) can be granted to a role or user. The **GRANT** command can be used to grant permissions to a user or role. These permissions granted are added to the existing ones.
- Permissions

  Grant a role's or user's permissions to one or more roles or users. 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 member can in turn grant permissions in the role to others, and revoke permissions in the role as well. If a role or user granted with certain permissions is changed or revoked, the permissions inherited from the role or user also change.

  A database administrator can grant permissions to and revoke them from any role or user. Roles having **CREATEROLE** permission can grant or revoke membership in any role that is not an administrator.

Hierarchical Permission Management

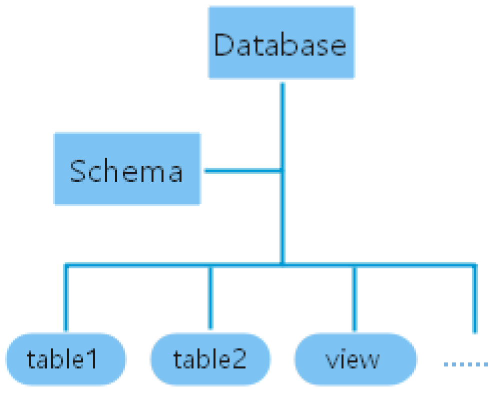
GaussDB(DWS) implements a hierarchical permission management on databases, schemas, and data objects.

- Databases cannot communicate with each other and share very few resources. Their connections and permissions can be isolated. The database cluster has one or more named databases. Users and roles are shared within the entire cluster, 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.
- Schemas share more resources than databases do. User permissions on schemas and subordinate objects can be flexibly configured using the **GRANT** and **REVOKE** syntax. Each database has one or more schemas. Each schema contains various types of objects, such as tables, views, and functions. To

access an object contained in a specified schema, a user must have the **USAGE** permission on the schema.

- After an object is created, by default, only the object owner or system administrator can query, modify, and delete the object. To access a specific database object, for example, **table1**, other users must be granted the **CONNECT** permission of database, the **USAGE** permission of schema, and the **SELECT** permission of **table1**. To access an object at the bottom layer, a user must be granted the permission on the object at the upper layer. To create or delete a schema, you must have the **CREATE** permission on its database.

Figure 5-3 Hierarchical Permission Management



Roles

The permission management model of GaussDB(DWS) is a typical implementation of the role-based permission control (RBAC). It manages users, roles, and permissions through this model.

A role is a set of permissions.

- The concept of "user" is equivalent to that of "role". The only difference is that "user" has the **login** permission while "role" has the **nologin** permission.
- Roles are assigned with different permissions based on their responsibilities in the database system. A role is a set of database permissions and represents the behavior constraints of a database user or a group of data users.
- Roles and users can be converted. You can use **ALTER** to assign the **login** permission to a role.
- 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 permission changes take effect for all the members of the role.
- In non-separation-of-duty scenarios, a role can be created, modified, and deleted only by a system administrator or a user with the **CREATEROLE** attribute. In separation-of-duty scenarios, a role can be created, modified, and deleted only by a user with the **CREATEROLE** attribute.

To view all roles, query the system catalog **PG_ROLES**.

```
SELECT * FROM PG_ROLES;
```

For how to create, modify, and delete a role, see "CREATE ROLE/ALTER ROLE/DROP ROLE" in *SQL Syntax Reference*.

Preset Roles

GaussDB(DWS) provides a group of preset roles. Their names start with **gs_role_**. These roles allow access to operations that require high permissions. You can grant these roles to other users or roles in the database for them to access or use specific information and functions. Exercise caution and ensure security when using preset roles.

The following table describes the permissions of preset roles.

Table 5-4 Permissions of preset roles

Role	Permission
gs_role_signal_backend	Invokes functions such as **pg_cancel_backend**, **pg_terminate_backend**, **pg_terminate_query**, **pg_cancel_query**, **pgxc_terminate_query**, and **pgxc_cancel_query** to cancel or terminate sessions, excluding those of the initial users.
gs_role_read_all_stats	Reads the system status view and uses various extension-related statistics, including information that is usually visible only to system administrators. For example: Resource management views: ● pgxc_wlm_operator_history ● pgxc_wlm_operator_info ● pgxc_wlm_operator_statistics ● pgxc_wlm_session_info ● pgxc_wlm_session_statistics ● pgxc_wlm_workload_records ● pgxc_workload_sql_count ● pgxc_workload_sql_elapse_time ● pgxc_workload_transaction Status information views: ● pgxc_stat_activity ● pgxc_get_table_skewness ● table_distribution ● pgxc_total_memory_detail ● pgxc_os_run_info ● pg_nodes_memory ● pgxc_instance_time ● pgxc_redo_stat

Role	Permission
gs_role_analyze_any	A user with the system-level **ANALYZE** permission can skip the schema permission check and perform **ANALYZE** on all tables.
gs_role_vacuum_any	A user with the system-level **VACUUM** permission can skip the schema permission check and perform **ANALYZE** on all tables.

Restrictions on using preset roles:

- **gs_role_** is the name field dedicated to preset roles in the database. Do not create users or roles starting with **gs_role_** or rename existing users or roles starting with **gs_role_**.
- Do not perform **ALTER** or **DROP** operations on preset roles.
- By default, a preset role does not have the **LOGIN** permission, so there is no preset login password for the role.
- The gsql meta-commands **\du** and **\dg** do not display information about preset roles. However, if **PATTERN** is specified, information about preset roles will be displayed.
- If the separation of permissions is disabled, the system administrator and users with the **ADMIN OPTION** permission of preset roles are allowed to perform GRANT and REVOKE operations on preset roles. If the separation of permissions is enabled, the security administrator (with the **CREATEROLE** attribute) and users with the **ADMIN OPTION** permission of preset roles are allowed to perform GRANT and REVOKE operations on preset roles. Example:
```
GRANT gs_role_signal_backend TO user1;
REVOKE gs_role_signal_backend FROM user1;
```

Granting or Revoking Permissions

A user who creates an object is the owner of this object. By default, **Separation of Duties in GaussDB(DWS)** is disabled after cluster 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 a user to use an object, the object owner or administrator can run the **GRANT** or **REVOKE** command to grant permissions to or revoke permissions from the user or role.

- Run the **GRANT** statement to grant permissions.

  For example, grant the permission of schema **myschema** to role **u1**, and grant the **SELECT** permission of table **myschema.t1** to role **u1**.
```
GRANT USAGE ON SCHEMA myschema TO u1;
GRANT SELECT ON TABLE myschema.t1 to u1;
```
- Run the **REVOKE** command to revoke a permission that has been granted.

  For example, revoke all permissions of user **u1** on the **myschema.t1** table.
```
REVOKE ALL PRIVILEGES ON myschema.t1 FROM u1;
```

5.1.5 Separation of Duties in GaussDB(DWS)

By default, the system administrator with the **SYSADMIN** attribute has the highest permission in the system. To avoid risks caused by centralized permissions, you can enable the separation of permissions to delegate system administrator permissions to security administrators and audit administrators.

- After the separation of permissions is enabled, a system administrator does not have the **CREATEROLE** attribute (security administrator) and **AUDITADMIN** attribute (audit administrator). That is, you do not have the permissions for creating roles and users and the permissions for viewing and maintaining database audit logs. For details about the **CREATEROLE** and **AUDITADMIN** attributes, see CREATE ROLE.
- After the separation of permissions is enabled, system administrators have the permissions only for the objects owned by them.

For how to configure permission separation, see **Configuring Separation of Permissions**

For details about permission changes before and after enabling the separation of permissions, see **Table 5-5** and **Table 5-6**.

Table 5-5 Default user permissions

Object	System Administrator	Security Administrator	Audit Administrator	Common User
Tablespace	Can create, modify, delete, access, and allocate tablespaces.	Cannot create, modify, delete, or allocate tablespaces, with authorization required for accessing tablespaces.		
Table	Has permissions for all tables.	Has permissions for its own tables, but does not have permissions for other users' tables.		
Index	Can create indexes on all tables.	Can create indexes on their own tables.		
Schema	Has permissions for all schemas.	Has all permissions for its own schemas, but does not have permissions for other users' schemas.		
Function	Has permissions for all functions.	Has permissions for its own functions, has the call permission for other users' functions in the **public** schema, but does not have permissions for other users' functions in other schemas.		
Customized view	Has permissions for all views.	Has permissions for its own views, but does not have permissions for other users' views.		

Object	System Administrator	Security Administrator	Audit Administrator	Common User
System catalog and system view	Has permissions for querying all system catalogs and views.	Has permissions for querying only some system catalogs and views. For details, see **GaussDB(DWS) System Catalogs and Views**.		

Table 5-6 Changes in permissions after the separation of permissions

Object	System Administrator	Security Administrator	Audit Administrator	Common User
Tablespace	No change	No change		
Table	Permissions reduced Has all permissions for its own tables, but does not have permissions for other users' tables in their schemas.	No change		
Index	Permissions reduced Can create indexes on its own tables.	No change		
Schema	Permissions reduced Has all permissions for its own schemas, but does not have permissions for other users' schemas.	No change		
Function	Permissions reduced Has all permissions for its own functions, but does not have permissions for other users' functions in their schemas.	No change		
Customized view	Permissions reduced Has all permissions for its own views and other users' views in the **public** schema, but does not have permissions for other users' views in their schemas.	No change		

Object	System Administrator	Security Administrator	Audit Administrator	Common User
System catalog and system view	No change	No change	No change	Has no permission for viewing any system catalogs or views.

5.2 GaussDB(DWS) Sensitive Data Management

5.2.1 GaussDB(DWS) Row-Level Access Control

The row-level access control feature restricts users to access only specific data rows in the data table, ensuring data read and write security.

Configuring Row-Level Access Control

Row-level access control 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**.

- You can use the **CREATE ROW LEVEL SECURITY POLICY** statement to create a row-level security policy on a table.

  This policy works only for expressions that take effect for specific database users and SQL operations. When a database user accesses the data table, if a SQL statement meets the specified row-level access control 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.
- After a row-level access control policy is created for a table, it takes effect only when the row-level access control switch (**ALTER TABLE**...**ENABLE ROW LEVEL SECURITY**) of the table is turned on.

Example of Row-Level Access Control

The data of all users is aggregated in table **all_data**. Implement row-level access control on this table so that different users can view only their own data.

Step 1 Create users **alice**, **bob**, and **peter**.

```
CREATE ROLE alice PASSWORD '********';
CREATE ROLE bob PASSWORD '********';
CREATE ROLE peter PASSWORD '********';
```

Create table **all_data** and insert data of different users into it.

```
CREATE TABLE public.all_data(id int, role varchar(100), data varchar(100));

INSERT INTO all_data VALUES(1, 'alice', 'alice data');
INSERT INTO all_data VALUES(2, 'bob', 'bob data');
INSERT INTO all_data VALUES(3, 'peter', 'peter data');
```

Step 2 Grant the read permission on table **all_data** to users **alice**, **bob**, and **peter**.

```
GRANT SELECT ON all_data TO alice, bob, peter;
```

Step 3 Run the **ALTER TABLE** *tablename* **ENABLE ROW LEVEL SECURITY** statement to enable the row-level access control.

```
ALTER TABLE all_data ENABLE ROW LEVEL SECURITY;
```

Step 4 Run the **CREATE ROW LEVEL SECURITY POLICY** statement to create a row-level access control policy so that the current user can view only its own data.

```
CREATE ROW LEVEL SECURITY POLICY all_data_rls ON all_data USING(role = CURRENT_USER);
```

Step 5 View information about the **all_data** table.

```
 \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
Distribute By: ROUND ROBIN
Location Nodes: ALL DATANODES
Options: orientation=row, compression=no, enable_rowsecurity=true
```

Step 6 Switch to user **alice** and query the data in table **all_data**. The query result shows that the row-level access control policy takes effect. User **alice** can only view its own data.

```
SET ROLE alice PASSWORD '********';

SELECT * FROM all_data;
 id | role  |    data
----+-------+------------
  1 | alice | alice data
```

The execution plan of the query is displayed, indicating that access to table **all_data** is under the row-level access control.

```
EXPLAIN(COSTS OFF) SELECT * FROM all_data;
                           QUERY PLAN
----------------------------------------------------------------
  id |          operation
 ----+------------------------------
   1 | ->  Streaming (type: GATHER)
   2 |    ->  Seq Scan on all_data

         Predicate Information (identified by plan id)
 --------------------------------------------------------------
   2 --Seq Scan on all_data
         Filter: ((role)::name = 'alice'::name)
 Notice: This query is influenced by row level security feature
```

```
  ====== Query Summary =====
 -------------------------------
 System available mem: 4833280KB
 Query Max mem: 4833280KB
 Query estimated mem: 1024KB
(16 rows)
```

Step 7 Switch to user **peter** and query the data in table **all_data**. The query result shows that the row-level access control policy takes effect. User **peter** can only view its own data.

```
SET ROLE peter PASSWORD '********';

SELECT * FROM all_data;
 id | role  |    data
----+-------+------------
  3 | peter | peter data
(1 row)
```

The execution plan of the table query is displayed, indicating that the query of table **all_data** is under the row-level access control.

```
EXPLAIN(COSTS OFF) SELECT * FROM all_data;
                          QUERY PLAN
----------------------------------------------------------------
  id |          operation
 ----+------------------------------
   1 | ->  Streaming (type: GATHER)
   2 |    ->  Seq Scan on all_data

        Predicate Information (identified by plan id)
 --------------------------------------------------------------
   2 --Seq Scan on all_data
         Filter: ((role)::name = 'peter'::name)
 Notice: This query is influenced by row level security feature

   ====== Query Summary =====
 -------------------------------
 System available mem: 4833280KB
 Query Max mem: 4833280KB
 Query estimated mem: 1024KB
(16 rows)
```

----End

5.2.2 GaussDB(DWS) Data Masking

GaussDB(DWS) provides the column-level dynamic data masking (DDM) function. For sensitive data (such as the ID card number, mobile number, and bank card number), the DDM function is used to redact the original data to protect data security and user privacy.

- Creating a data masking policy for a table

  GaussDB(DWS) uses the **CREATE REDACTION POLICY** syntax to create a data masking policy on a table. (**MASK_NONE**: Do not perform masking. **MASK_FULL**: Mask data into a fixed value. **MASK_PARTIAL**: Perform partial masking based on the character type, numeric type, or time type.)
- Modifying the data masking policy of a table

  The **ALTER REDACTION POLICY** syntax is used to modify the expression for enabling a masking policy, rename a masking policy, and add, modify, or delete masked columns.
- Deleting the masking policy of a table

The **DROP REDACTION POLICY** syntax is used to delete the masking function information of a masking policy on all columns of a table.

- Viewing the masking policy and masked columns

  Redaction policy information is stored in the system catalog **PG_REDACTION_POLICY**, and redacted column information is stored in the system catalog **PG_REDACTION_COLUMN**. You can view information about the redaction policy and redacted columns in the system views **REDACTION_POLICIES** and **REDACTION_COLUMNS**.

NOTE

- Generally, you can run the SELECT statement to view the data redaction result. If a statement has the following features, sensitive data may be deliberately obtained. In this case, an error will be reported during statement execution.
  - The GROUP BY clause references the Target Entry containing redaction columns as the target column.
  - DISTINCT works on the output redaction columns.
  - The statement contains CTE.
  - Operations on sets are involved.
  - The target columns of a subquery are not redaction columns of the base table, but the expressions or function calls for redaction columns of the base table.
- You can use COPY TO or GDS to export the redacted data. Due to the irreversibility of the data redaction, secondary redaction of the data is meaningless.
- Do not set target columns of UPDATE, MERGE INTO, and DELETE statements to redaction columns.
- The UPSERT statement allows you to insert update data through EXCLUDED. If data in the base table is updated by referencing redaction columns, the data may be modified by mistake. As a result, an error will be reported during the execution.

Examples

The following uses the employee table **emp**, table owner **alice**, and roles **matu** and **july** as an example to illustrate the data masking process. The **emp** table contains private data such as the employee name, mobile number, email address, bank card number, and salary.

Step 1 After connecting to the database as the administrator, create roles **alice**, **matu**, and **july**.

```
CREATE ROLE alice PASSWORD '{Password}';
CREATE ROLE matu PASSWORD '{Password}';
CREATE ROLE july PASSWORD '{Password}';
```

Step 2 Grant schema permissions on the current database to **alice**, **matu**, and **july**.

```
GRANT ALL PRIVILEGES on schema public to alice,matu,july;
```

Step 3 Switch to role **alice**, create the **emp** table, and insert three pieces of employee information.

```
SET ROLE alice PASSWORD '{Password}';

CREATE TABLE emp(id int, name varchar(20), phone_no varchar(11), card_no number, card_string
varchar(19), email text, salary numeric(100, 4), birthday date);

INSERT INTO emp VALUES(1, 'anny', '13420002340', 1234123412341234, '1234-1234-1234-1234',
'smithWu@163.com', 10000.00, '1999-10-02');
INSERT INTO emp VALUES(2, 'bob', '18299023211', 3456345634563456, '3456-3456-3456-3456',
'66allen_mm@qq.com', 9999.99, '1989-12-12');
```

```
INSERT INTO emp VALUES(3, 'cici', '15512231233', NULL, NULL, 'jonesishere@sina.com', NULL,
'1992-11-06');
```

Step 4 **alice** grants the read permission on the **emp** table to **matu** and **july**.

```
GRANT SELECT ON emp TO matu, july;
```

Step 5 Create the masking policy **mask_emp**: Only user **alice** can view all employee information. User **matu** and **july** cannot view employee bank card numbers and salary data. The **card_no** column is of the numeric type and all of its data is masked into 0 by the **MASK_FULL** function. The **card_string** column is of the character type and part of its data is masked by the **MASK_PARTIAL** function based on the specified input and output formats. The **salary** column is of the numeric type and the **MASK_PARTIAL** function is used to mask all digits before the penultimate digit using the number 9.

```
CREATE REDACTION POLICY mask_emp ON emp WHEN (current_user IN ('matu', 'july'))
 ADD COLUMN card_no WITH mask_full(card_no),
 ADD COLUMN card_string WITH mask_partial(card_string, 'VVVVFVVVVFVVVVFVVVV','VVVV-VVVV-VVVV-
VVVV','#',1,12),
 ADD COLUMN salary WITH mask_partial(salary, '9', 1, length(salary) - 2);
```

Step 6 Switch to **matu** and **july** and view the employee table **emp**.

```
SET ROLE matu PASSWORD '{Password}';
SELECT * FROM emp;
 id | name |  phone_no   | card_no |     card_string     |        email         |   salary   |      birthday
----+------+-------------+---------+---------------------+----------------------+------------+---------------------
  1 | anny | 13420002340 |       0 | ####-####-####-1234 | smithWu@163.com      | 99999.9990 |
1999-10-02 00:00:00
  2 | bob  | 18299023211 |       0 | ####-####-####-3456 | 66allen_mm@qq.com    |  9999.9990 |
1989-12-12 00:00:00
  3 | cici | 15512231233 |         |                     | jonesishere@sina.com |            | 1992-11-06 00:00:00
(3 rows)

SET ROLE july PASSWORD '{Password}';
SELECT * FROM emp;
 id | name |  phone_no   | card_no |     card_string     |        email         |   salary   |      birthday
----+------+-------------+---------+---------------------+----------------------+------------+---------------------
  1 | anny | 13420002340 |       0 | ####-####-####-1234 | smithWu@163.com      | 99999.9990 |
1999-10-02 00:00:00
  2 | bob  | 18299023211 |       0 | ####-####-####-3456 | 66allen_mm@qq.com    |  9999.9990 |
1989-12-12 00:00:00
  3 | cici | 15512231233 |         |                     | jonesishere@sina.com |            | 1992-11-06 00:00:00
(3 rows)
```

Step 7 If you want **matu** to have the permission to view all employee information, but do not want **july** to have. In this case, you only need to modify the effective scope of the policy.

```
SET ROLE alice PASSWORD '{Password}';
ALTER REDACTION POLICY mask_emp ON emp WHEN(current_user = 'july');
```

Step 8 Switch to users **matu** and **july** and view the **emp** table again, respectively.

```
SET ROLE matu PASSWORD '{Password}';
SELECT * FROM emp;
 id | name |  phone_no   |     card_no      |     card_string     |        email         |   salary   |      birthday
----+------+-------------+------------------+---------------------+----------------------+------------
+---------------------
  1 | anny | 13420002340 | 1234123412341234 | 1234-1234-1234-1234 | smithWu@163.com      |
10000.0000 | 1999-10-02 00:00:00
  2 | bob  | 18299023211 | 3456345634563456 | 3456-3456-3456-3456 | 66allen_mm@qq.com    |
9999.9900 | 1989-12-12 00:00:00
  3 | cici | 15512231233 |                  |                     | jonesishere@sina.com |            | 1992-11-06 00:00:00
(3 rows)

SET ROLE july PASSWORD '{Password}';
SELECT * FROM emp;
 id | name |  phone_no   | card_no |     card_string     |        email         |   salary   |      birthday
```

```
----+------+-------------+---------+---------------------+----------------------+------------+---------------------
  1 | anny | 13420002340 |       0 | ####-####-####-1234 | smithWu@163.com      | 99999.9990 |
1999-10-02 00:00:00
  2 | bob  | 18299023211 |       0 | ####-####-####-3456 | 66allen_mm@qq.com    |  9999.9990 |
1989-12-12 00:00:00
  3 | cici | 15512231233 |         |                     | jonesishere@sina.com |            | 1992-11-06 00:00:00
(3 rows)
```

Step 9 The information in the **phone_no**, **email**, and **birthday** columns is private data. Update redaction policy **mask_emp** and add three redaction columns.

```
SET ROLE alice PASSWORD '{Password}';
ALTER REDACTION POLICY mask_emp ON emp ADD COLUMN phone_no WITH mask_partial(phone_no, '*',
4);
ALTER REDACTION POLICY mask_emp ON emp ADD COLUMN email WITH mask_partial(email, '*', 1,
position('@' in email));
ALTER REDACTION POLICY mask_emp ON emp ADD COLUMN birthday WITH mask_full(birthday);
```

Step 10 Switch to **july** and view data in the **emp** table.

```
SET ROLE july PASSWORD '{Password}';
SELECT * FROM emp;
 id | name |  phone_no   | card_no |     card_string     |        email         |   salary   |      birthday
----+------+-------------+---------+---------------------+----------------------+------------+---------------------
  1 | anny | 134******** |       0 | ####-####-####-1234 | ********163.com      | 99999.9990 | 1970-01-01
00:00:00
  2 | bob  | 182******** |       0 | ####-####-####-3456 | ***********qq.com    |  9999.9990 | 1970-01-01
00:00:00
  3 | cici | 155******** |         |                     | ************sina.com |            | 1970-01-01 00:00:00
(3 rows)
```

Step 11 Query **redaction_policies** and **redaction_columns** to view details about the current redaction policy **mask_emp**.

```
SELECT * FROM redaction_policies;
 object_schema | object_owner | object_name | policy_name |            expression             | enable |
policy_description | inherited
---------------+--------------+-------------+-------------+-----------------------------------+--------
+--------------------+-----------
 public        | alice        | emp         | mask_emp    | ("current_user"() = 'july'::name) | t      |                    |
f
(1 row)

SELECT object_name, column_name, function_info FROM redaction_columns;
 object_name | column_name |                                                 function_info
-------------+-------------
+-------------------------------------------------------------------------------------------------------
 emp         | card_no     | mask_full(card_no)
 emp         | card_string | mask_partial(card_string, 'VVVVFVVVVFVVVVFVVVV'::text, 'VVVV-VVVV-VVVV-
VVVV'::text, '#'::text, 1, 12)
 emp         | email       | mask_partial(email, '*'::text, 1, "position"(email, '@'::text))
 emp         | salary      | mask_partial(salary, '9'::text, 1, (length((salary)::text) - 2))
 emp         | birthday    | mask_full(birthday)
 emp         | phone_no    | mask_partial(phone_no, '*'::text, 4)
(6 rows)
```

Step 12 Add the **salary_info** column. To replace the salary information in text format with *.*, you can create a user-defined redaction function. In this step, you can use the PL/pgSQL to define the redaction function **mask_regexp_salary**. To create a redaction column, you simply need to customize the function name and parameter list. For details, see **GaussDB(DWS) User-Defined Functions**.

```
SET ROLE alice PASSWORD '{Password}';

ALTER TABLE emp ADD COLUMN salary_info TEXT;
UPDATE emp SET salary_info = salary::text;

CREATE FUNCTION mask_regexp_salary(salary_info text) RETURNS text AS
$$
 SELECT regexp_replace($1, '[0-9]+','*','g');
```

```
$$
 LANGUAGE SQL
STRICT SHIPPABLE;

ALTER REDACTION POLICY mask_emp ON emp ADD COLUMN salary_info WITH
mask_regexp_salary(salary_info);

SET ROLE july PASSWORD '{Password}';
SELECT id, name, salary_info FROM emp;
 id | name | salary_info
----+------+-------------
  1 | anny | *.*
  2 | bob  | *.*
  3 | cici |
(3 rows)
```

Step 13 If there is no need to set a redaction policy for the **emp** table, delete redaction policy **mask_emp**.

```
SET ROLE alice PASSWORD '{Password}';
DROP REDACTION POLICY mask_emp ON emp;
```

----End

5.2.3 Encrypting and Decrypting GaussDB(DWS) Strings

GaussDB(DWS) supports encryption and decryption of strings using the following functions:

- gs_encrypt(encryptstr, keystr, cryptotype, cryptomode, hashmethod)

  Description: Encrypts an **encryptstr** string using the **keystr** key based on the encryption algorithm specified by **cryptotype** and **cryptomode** and the HMAC algorithm specified by **hashmethod**, and returns the encrypted string. **cryptotype** can be **aes128**, **aes192**, **aes256**, or **sm4**. **cryptomode** is **cbc**. **hashmethod** can be **sha256**, **sha384**, **sha512**, or **sm3**. Currently, the following types of data can be encrypted: numerals supported in the database; character type; RAW in binary type; and DATE, TIMESTAMP, and SMALLDATETIME in date/time type. The **keystr** length is related to the encryption algorithm and contains 1 to **KeyLen** bytes. If **cryptotype** is **aes128** or **sm4**, **KeyLen** is **16**; if **cryptotype** is **aes192**, **KeyLen** is **24**; if **cryptotype** is **aes256**, **KeyLen** is **32**.

  Return type: text

  Length of the return value: at least 4 x [(maclen + 56)/3] bytes and no more than 4 x [(Len + maclen + 56)/3] bytes, where **Len** indicates the string length (in bytes) before the encryption and **maclen** indicates the length of the HMAC value. If **hashmethod** is **sha256** or **sm3**, **maclen** is **32**; if **hashmethod** is **sha384**, **maclen** is **48**; if **hashmethod** is **sha512**, **maclen** is **64**. That is, if **hashmethod** is **sha256** or **sm3**, the returned string contains 120 to 4 x [(Len + 88)/3] bytes; if **hashmethod** is **sha384**, the returned string contains 140 to 4 x [(Len + 104)/3] bytes; if **hashmethod** is **sha512**, the returned string contains 160 to 4 x [(Len + 120)/3] bytes.

  Example:

  ```
  SELECT gs_encrypt('GaussDB(DWS)', '1234', 'aes128', 'cbc',  'sha256');
                                                gs_encrypt
  ------------------------------------------------------------------------------------------------------------------------
  ---
   AAAAAAAAAACcFjDcCSbop7D87sOa2nxTFrkE9RJQGK34ypgrOPsFJIqggI8tl
  +eMDcQYT3po98wPCC7VBfhv7mdBy7IVnzdrp0rdMrD6/zTl8w0v9/s2OA==
  (1 row)
  ```

NOTE

- A decryption password is required during the execution of this function. For security purposes, the gsql tool does not record this function in the execution history. That is, the execution history of this function cannot be found in **gsql** by paging up and down.
- Do not use the **ge_encrypt** and **gs_encrypt_aes128** functions for the same data table.

- gs_decrypt(decryptstr, keystr,cryptotype, cryptomode, hashmethod)

  Description: Decrypts a **decryptstr** string using the **keystr** key based on the encryption algorithm specified by **cryptotype** and **cryptomode** and the HMAC algorithm specified by **hashmethod**, and returns the decrypted string. The **keystr** used for decryption must be consistent with that used for encryption. **keystr** cannot be empty.

  Return type: text

  Example:

```
SELECT gs_decrypt('AAAAAAAAAACcFjDcCSbop7D87sOa2nxTFrkE9RJQGK34ypgrOPsFJIqggI8tl
+eMDcQYT3po98wPCC7VBfhv7mdBy7IVnzdrp0rdMrD6/zTl8w0v9/s2OA==', '1234', 'aes128', 'cbc',
'sha256');
  gs_decrypt
--------------
 GaussDB(DWS)
(1 row)
```

NOTE

- A decryption password is required during the execution of this function. For security purposes, the gsql tool does not record this function in the execution history. That is, the execution history of this function cannot be found in **gsql** by paging up and down.
- This function works with the **gs_encrypt** function, and the two functions must use the same encryption algorithm and HMAC algorithm.

- gs_encrypt_aes128(encryptstr,keystr)

  Description: Encrypts **encryptstr** strings using **keystr** as the key and returns encrypted strings. The length of **keystr** ranges from 1 to 16 bytes. Currently, the following types of data can be encrypted: numerals supported in the database; character type; RAW in binary type; and DATE, TIMESTAMP, and SMALLDATETIME in date/time type.

  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:

```
SELECT gs_encrypt_aes128('DWS','1234');
                              gs_encrypt_aes128
-------------------------------------------------------------------------------------------
 MGFX/AvA69PvS6wgZMtEAwNdjf/lMM6b7pIY5miAAkS0cf3m5mKl8iNe1BKDVqTvgZEEoMTycVVE
+tHF69uHYznXyhs=
(1 row)
```

NOTE

- A decryption password is required during the execution of this function. For security purposes, the gsql tool does not record this function in the execution history. That is, the execution history of this function cannot be found in **gsql** by paging up and down.
- Do not use the **ge_encrypt** and **gs_encrypt_aes128** functions for the same data table.

- gs_decrypt_aes128(decryptstr,keystr)

  Description: Decrypts a **decryptstr** string using the **keystr** key and returns the decrypted string. The **keystr** used for decryption must be consistent with that used for encryption. **keystr** cannot be empty.

  Return type: text

  Example:

```
SELECT gs_decrypt_aes128('MGFX/AvA69PvS6wgZMtEAwNdjf/
lMM6b7pIY5miAAkS0cf3m5mKl8iNe1BKDVqTvgZEEoMTycVVE+tHF69uHYznXyhs=','1234');
 gs_decrypt_aes128
-------------------
 DWS
(1 row)
```

  NOTE

  - A decryption password is required during the execution of this function. For security purposes, the gsql tool does not record this function in the execution history. That is, the execution history of this function cannot be found in **gsql** by paging up and down.
  - This function works with the **gs_encrypt_aes128** function.

- gs_hash(hashstr, hashmethod)

  Description: Obtains the digest string of a **hashstr** string based on the algorithm specified by **hashmethod**. **hashmethod** can be **sha256**, **sha384**, **sha512**, or **sm3**.

  Return type: text

  Length of the return value: 64 bytes if **hashmethod** is **sha256** or **sm3**; 96 bytes if **hashmethod** is **sha384**; 128 bytes if **hashmethod** is **sha512**

  Example:

```
SELECT gs_hash('GaussDB(DWS)', 'sha256');
                                          gs_hash
--------------------------------------------------------------------------------------------------

e59069daa6541ae20af7c747662702c731b26b8abd7a788f4d15611aa0db608efdbb5587ba90789a983f8
5dd51766609
(1 row)
```

- md5(string)

  Description: Encrypts a string in MD5 mode and returns a value in hexadecimal form.

  NOTE

  MD5 is insecure and is not recommended.

  Return type: text

  Example:

```
SELECT md5('ABC');
             md5
```

```
----------------------------------
 902fbdd2b1df0c4f70b4a5d23525e932
(1 row)
```

6 GaussDB(DWS) Data Query

6.1 GaussDB(DWS) Single-Table Query

Example table:

```
CREATE TABLE newproducts
(
product_id INTEGER NOT NULL,
product_name VARCHAR2(60),
category VARCHAR2(60),
quantity  INTEGER
)
WITH (ORIENTATION = COLUMN) DISTRIBUTE BY HASH(product_id);


INSERT INTO newproducts VALUES (1502, 'earphones', 'electronics',150);
INSERT INTO newproducts VALUES (1601, 'telescope', 'toys',80);
INSERT INTO newproducts VALUES (1666, 'Frisbee', 'toys',244);
INSERT INTO newproducts VALUES (1700, 'interface', 'books',100);
INSERT INTO newproducts VALUES (2344, 'milklotion', 'skin care',320);
INSERT INTO newproducts VALUES (3577, 'dumbbell', 'sports',550);
INSERT INTO newproducts VALUES (1210, 'necklace', 'jewels', 200);
```

Simple Queries

Run the **SELECT... FROM...** statement to obtain the result from the database.

```
SELECT category FROM newproducts;
  category
------------
 electr
 sports
 jewels
 toys
 books
 skin care
 toys
(7 rows)
```

Filtering Test Results

Run the **WHERE** statement to filter the query result and find the queried part.

```
SELECT * FROM newproducts WHERE category='toys';
 product_id | product_name | category | quantity
```

```
------------+--------------+----------+----------
       1601 | telescope    | toys     |       80
       1666 | Frisbee      | toys     |      244
(2 rows)
```

Sorting Results

Use the **ORDER BY** statement to sort query results.

```
SELECT product_id,product_name,category,quantity FROM newproducts ORDER BY quantity DESC;
 product_id | product_name |  category   | quantity
------------+--------------+-------------+----------
       3577 | dumbbell     | sports      |      550
       2344 | milklotion   | skin care   |      320
       1666 | Frisbee      | toys        |      244
       1210 | necklace     | jewels      |      200
       1502 | earphones    | electronics |      150
       1700 | interface    | books       |      100
       1601 | telescope    | toys        |       80
(7 rows)
```

Limiting the Number of Query Results

If you want the query to return only part of the result, you can use the **LIMIT** statement to limit the number of records returned in the query result.

```
SELECT product_id,product_name,category,quantity FROM newproducts ORDER BY quantity DESC limit 5;
 product_id | product_name |  category   | quantity
------------+--------------+-------------+----------
       3577 | dumbbell     | sports      |      550
       2344 | milklotion   | skin care   |      320
       1666 | Frisbee      | toys        |      244
       1210 | necklace     | jewels      |      200
       1502 | earphones    | electronics |      150
(5 rows)
```

Aggregated Query

If you want query data comprehensively, you can use the **GROUP BY** statement and aggregate functions to construct an aggregated query.

```
SELECT category, string_agg(quantity,',') FROM newproducts group by category;
  category   | string_agg
-------------+------------
 toys        | 80,244
 books       | 100
 sports      | 550
 jewels      | 200
 skin care   | 320
 electronics | 150
```

6.2 GaussDB(DWS) Multi-Table Join Query

Join Types

Multiple joins are necessary for accomplishing complex queries. Joins are classified into inner joins and outer joins. Each type of joins have their subtypes.

- Inner join: inner join, cross join, and natural join.
- Outer join: left outer join, right outer join, and full join.

To better illustrate the differences between these joins, the following provides some examples.

Create the sample tables **student** and **math_score** and insert data into them. Set **enable_fast_query_shipping** to **off** (**on** by default), that is, the query optimizer uses the distributed framework. Set **explain_perf_mode** to **pretty** (default value) to specify the **EXPLAIN** display format.

```
CREATE TABLE student(
  id INTEGER,
  name varchar(50)
);

CREATE TABLE math_score(
  id INTEGER,
  score INTEGER
);

INSERT INTO student VALUES(1, 'Tom');
INSERT INTO student VALUES(2, 'Lily');
INSERT INTO student VALUES(3, 'Tina');
INSERT INTO student VALUES(4, 'Perry');

INSERT INTO math_score VALUES(1, 80);
INSERT INTO math_score VALUES(2, 75);
INSERT INTO math_score VALUES(4, 95);
INSERT INTO math_score VALUES(6, NULL);

SET enable_fast_query_shipping = off;
SET explain_perf_mode = pretty;
```

Inner Join

- Inner join

  Syntax:

  ```
  left_table [INNER] JOIN right_table [ ON join_condition | USING ( join_column )]
  ```

  Description: Rows that meet **join_condition** in both the left and right tables are joined and output. Tuples that do not meet **join_condition** are not output.

  Example 1: Query students' math scores.

  ```
  SELECT s.id, s.name, ms.score FROM student s JOIN math_score ms on s.id = ms.id;
   id | name  | score
  ----+-------+-------
    2 | Lily  |    75
    1 | Tom   |    80
    4 | Perry |    95
  (3 rows)


  EXPLAIN SELECT s.id, s.name, ms.score FROM student s JOIN math_score ms on s.id = ms.id;
                                        QUERY PLAN
  -------------------------------------------------------------------------------------------
    id |                operation                | E-rows | E-memory | E-width | E-costs
   ----+-----------------------------------------+--------+----------+---------+---------
     1 | ->  Streaming (type: GATHER)            |      4 |          |      13 | 19.47
     2 |    ->  Hash Join (3,4)                  |      4 | 1MB      |      13 | 11.47
     3 |       ->  Seq Scan on math_score ms     |     30 | 1MB      |       8 | 10.10
     4 |       ->  Hash                          |     12 | 16MB     |       9 | 1.28
     5 |          ->  Streaming(type: BROADCAST) |     12 | 2MB      |       9 | 1.28
     6 |             ->  Seq Scan on student s   |      4 | 1MB      |       9 | 1.01

   Predicate Information (identified by plan id)
   ---------------------------------------------
     2 --Hash Join (3,4)
           Hash Cond: (ms.id = s.id)
  ```

```
  ====== Query Summary =====
 -------------------------------
 System available mem: 1761280KB
 Query Max mem: 1761280KB
 Query estimated mem: 4400KB
(19 rows)
```

- Cross join

  Syntax:

  ```
  left_table CROSS JOIN right_table
  ```

  Description: Each row in the left table is joined with each row in the right table. The number of final rows is the product of the number of rows on both sides. The product is also called Cartesian product.

  Example 2: Cross join of student tables and math score tables.

  ```
  SELECT s.id, s.name, ms.score FROM student s CROSS JOIN math_score ms;
   id | name  | score
  ----+-------+-------
    3 | Tina  |    80
    2 | Lily  |    80
    1 | Tom   |    80
    4 | Perry |    80
    3 | Tina  |
    2 | Lily  |
    1 | Tom   |
    4 | Perry |
    3 | Tina  |    95
    2 | Lily  |    95
    1 | Tom   |    95
    4 | Perry |    95
    2 | Lily  |    75
    3 | Tina  |    75
    1 | Tom   |    75
    4 | Perry |    75
  (16 rows)

  EXPLAIN SELECT s.id, s.name, ms.score FROM student s CROSS JOIN math_score ms;
                                  QUERY PLAN
  ------------------------------------------------------------------------------------------
    id |                operation                | E-rows | E-memory | E-width | E-costs
   ----+-----------------------------------------+--------+----------+---------+---------
     1 | ->  Streaming (type: GATHER)            |    120 |          |      13 | 19.89
     2 |    ->  Nested Loop (3,4)                |    120 | 1MB      |      13 | 11.89
     3 |       ->  Seq Scan on math_score ms     |     30 | 1MB      |       4 | 10.10
     4 |       ->  Materialize                   |     12 | 16MB     |       9 | 1.30
     5 |          ->  Streaming(type: BROADCAST) |     12 | 2MB      |       9 | 1.28
     6 |             ->  Seq Scan on student s   |      4 | 1MB      |       9 | 1.01

    ====== Query Summary =====
   -------------------------------
   System available mem: 1761280KB
   Query Max mem: 1761280KB
   Query estimated mem: 4144KB
  (14 rows)
  ```

- Natural join

  Syntax:

  ```
  left_table NATURAL JOIN right_table
  ```

  Description: Columns with the same name in left table and right table are joined by equi-join, and the columns with the same name are merged into one column.

  Example 3: Natural join between the **student** table and the **math_score** table. The columns with the same name in the two tables are the **id** columns, therefore equivalent join is performed based on the **id** columns.

```
SELECT * FROM student s NATURAL JOIN math_score ms;
 id | name  | score
----+-------+-------
  1 | Tom   |    80
  4 | Perry |    95
  2 | Lily  |    75
(3 rows)

EXPLAIN SELECT * FROM student s NATURAL JOIN math_score ms;
                                   QUERY PLAN
------------------------------------------------------------------------------------------
  id |                operation                | E-rows | E-memory | E-width | E-costs
 ----+-----------------------------------------+--------+----------+---------+---------
   1 | ->  Streaming (type: GATHER)            |      4 |          |      13 | 19.47
   2 |    ->  Hash Join (3,4)                  |      4 | 1MB      |      13 | 11.47
   3 |       ->  Seq Scan on math_score ms     |     30 | 1MB      |       8 | 10.10
   4 |       ->  Hash                          |     12 | 16MB     |       9 | 1.28
   5 |          ->  Streaming(type: BROADCAST) |     12 | 2MB      |       9 | 1.28
   6 |             ->  Seq Scan on student s   |      4 | 1MB      |       9 | 1.01

 Predicate Information (identified by plan id)
 ---------------------------------------------
   2 --Hash Join (3,4)
         Hash Cond: (ms.id = s.id)

   ====== Query Summary =====
 -------------------------------
 System available mem: 1761280KB
 Query Max mem: 1761280KB
 Query estimated mem: 4400KB
(19 rows)
```

Outer Join

- Left Join

  Syntax:

  ```
  left_table LEFT [OUTER] JOIN right_table [ ON join_condition | USING ( join_column )]
  ```

  Description: The result set of a left outer join includes all rows of left table, not only the joined rows. If a row in the left table does not match any row in right table, the row will be **NULL** in the result set.

  Example 4: Perform left join on the **student** table and **math_score** table. The right table data corresponding to the row where ID is 3 in the **student** table is filled with **NULL** in the result set.

  ```
  SELECT s.id, s.name, ms.score FROM student s LEFT JOIN math_score ms on (s.id = ms.id);
   id | name  | score
  ----+-------+-------
    3 | Tina  |
    1 | Tom   |    80
    2 | Lily  |    75
    4 | Perry |    95
  (4 rows)

  EXPLAIN SELECT s.id, s.name, ms.score FROM student s LEFT JOIN math_score ms on (s.id = ms.id);
                                     QUERY PLAN
  -------------------------------------------------------------------------------------------
    id |                 operation                  | E-rows | E-memory | E-width | E-costs
   ----+--------------------------------------------+--------+----------+---------+---------
     1 | ->  Streaming (type: GATHER)               |      4 |          |      13 | 10.26
     2 |    ->  Hash Left Join (3, 5)               |      4 | 1MB      |      13 | 2.26
     3 |       ->  Streaming(type: REDISTRIBUTE)    |      4 | 2MB      |       9 | 1.11
     4 |          ->  Seq Scan on student s         |      4 | 1MB      |       9 | 1.01
     5 |       ->  Hash                             |      4 | 16MB     |       8 | 1.11
     6 |          ->  Streaming(type: REDISTRIBUTE) |      4 | 2MB      |       8 | 1.11
     7 |             ->  Seq Scan on math_score ms  |      4 | 1MB      |       8 | 1.01
  ```

```
 Predicate Information (identified by plan id)
 ---------------------------------------------
   2 --Hash Left Join (3, 5)
         Hash Cond: (s.id = ms.id)

   ====== Query Summary =====
 -----------------------------
 System available mem: 901120KB
 Query Max mem: 901120KB
 Query estimated mem: 7520KB
(20 rows)
```

- Right join

  Syntax:

  ```
  left_table RIGHT [OUTER] JOIN right_table [ ON join_condition | USING ( join_column )]
  ```

  Description: Contrary to the left join, the result set of a right join includes all rows of the right table, not just the joined rows. If a row in the right table does not match any row in right table, the row will be **NULL** in the result set.

  Example 5: Perform right join on the **student** table and **math_score** table. The right table data corresponding to the row where ID is 6 in the **math_score** table is filled with **NULL** in the result set.

  ```
  SELECT ms.id, s.name, ms.score FROM student s RIGHT JOIN math_score ms on (s.id = ms.id);
   id | name  | score
  ----+-------+-------
    1 | Tom   |    80
    6 |       |
    4 | Perry |    95
    2 | Lily  |    75

  EXPLAIN SELECT ms.id, s.name, ms.score FROM student s RIGHT JOIN math_score ms on (s.id = ms.id);
                                    QUERY PLAN
  -----------------------------------------------------------------------------------------
    id |                operation                | E-rows | E-memory | E-width | E-costs
   ----+-----------------------------------------+--------+----------+---------+---------
     1 | ->  Streaming (type: GATHER)            |     30 |          |      13 | 19.47
     2 |    ->  Hash Left Join (3, 4)            |     30 | 1MB      |      13 | 11.47
     3 |       ->  Seq Scan on math_score ms     |     30 | 1MB      |       8 | 10.10
     4 |       ->  Hash                          |     12 | 16MB     |       9 | 1.28
     5 |          ->  Streaming(type: BROADCAST) |     12 | 2MB      |       9 | 1.28
     6 |             ->  Seq Scan on student s   |      4 | 1MB      |       9 | 1.01

   Predicate Information (identified by plan id)
   ---------------------------------------------
     2 --Hash Left Join (3, 4)
           Hash Cond: (ms.id = s.id)

     ====== Query Summary =====
   ------------------------------
   System available mem: 1761280KB
   Query Max mem: 1761280KB
   Query estimated mem: 5424KB
  (19 rows)
  ```

  In a right join, **Left** is displayed in the join operator. This is because a right join is actually the process replacing the left table with the right table then performing left join.

- Full join

  Syntax:

  ```
  left_table FULL [OUTER] JOIN right_table [ ON join_condition | USING ( join_column )]
  ```

  Description: A full join is a combination of a left outer join and a right outer join. The result set of a full outer join includes all rows of the left table and the right table, not just the joined rows. If a row in the left table does not match any row in the right table, the row will be **NULL** in the result set. If a row in the right table does not match any row in right table, the row will be **NULL** in the result set.

Example 6: Perform full outer join on the **student** table and **math_score** table. The right table data corresponding to the row where ID is 3 is filled with **NULL** in the result set. The left table data corresponding to the row where ID is 6 is filled with **NULL** in the result set.

```
SELECT s.id, s.name, ms.id, ms.score FROM student s FULL JOIN math_score ms ON (s.id = ms.id);
 id | name  | id | score
----+-------+----+-------
  2 | Lily  |  2 |    75
  4 | Perry |  4 |    95
  1 | Tom   |  1 |    80
  3 | Tina  |    |
    |       |  6 |
(5 rows)

EXPLAIN SELECT s.id, s.name, ms.id, ms.score FROM student s FULL JOIN math_score ms ON (s.id =
ms.id);
                                  QUERY PLAN
-------------------------------------------------------------------------------------------
  id |                 operation                  | E-rows | E-memory | E-width | E-costs
 ----+--------------------------------------------+--------+----------+---------+---------
   1 | ->  Streaming (type: GATHER)               |     30 |          |      17 | 20.24
   2 |    ->  Hash Full Join (3, 5)               |     30 | 1MB      |      17 | 12.24
   3 |       ->  Streaming(type: REDISTRIBUTE)    |     30 | 2MB      |       8 | 11.06
   4 |          ->  Seq Scan on math_score ms     |     30 | 1MB      |       8 | 10.10
   5 |       ->  Hash                             |      4 | 16MB     |       9 | 1.11
   6 |          ->  Streaming(type: REDISTRIBUTE) |      4 | 2MB      |       9 | 1.11
   7 |             ->  Seq Scan on student s      |      4 | 1MB      |       9 | 1.01

 Predicate Information (identified by plan id)
 --------------------------------------------
   2 --Hash Full Join (3, 5)
         Hash Cond: (ms.id = s.id)

   ====== Query Summary =====
 -------------------------------
 System available mem: 1761280KB
 Query Max mem: 1761280KB
 Query estimated mem: 6496KB
(20 rows)
```

Differences Between the ON Condition and the WHERE Condition in Multi-Table Query

According to the preceding join syntax, except natural join and cross join, the **ON** condition (**USING** is converted to the **ON** condition during query parsing) is used on the join result of both the two tables. Generally, the **WHERE** condition is used in the query statement to restrict the query result. The **ON** join condition and **WHERE** filter condition do not contain conditions that can be pushed down to tables. The differences between **ON** and **WHERE** are as follows:

- The **ON** condition is used for joining two tables.
- **WHERE** is used to filter the result set.

To sum up, the **ON** condition is used when two tables are joined. After the join result set of two tables is generated, the **WHERE** condition is used.

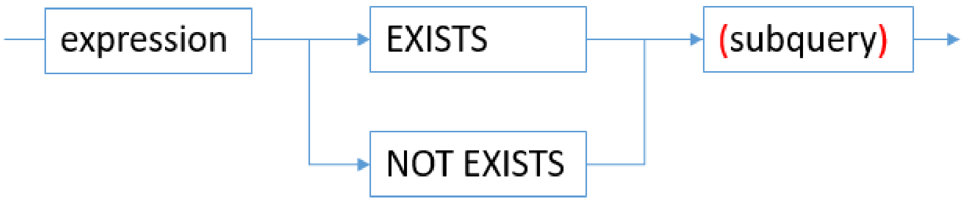
6.3 GaussDB(DWS) Subquery Expressions

A subquery allows you to nest one query within another, enabling more complex data query and analysis.

Subquery Expressions

- EXISTS/NOT EXISTS

  Before the main query runs, the subquery runs and its result determines if the main query continues. EXISTS returns **true** if the subquery returns at least one row. **NOT EXISTS** returns **true** if the subquery returns no rows.

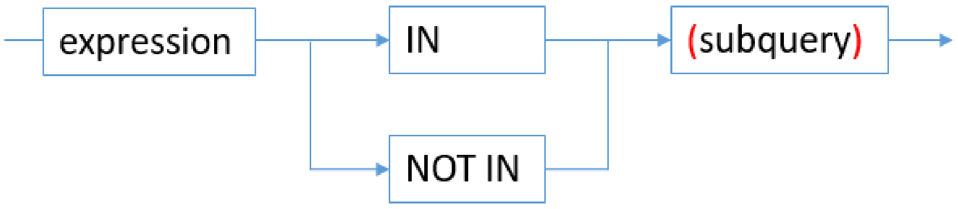
  

  Syntax:

  ```
  WHERE column_name EXISTS/NOT EXISTS (subquery)
  ```

- IN/NOT IN

  IN and NOT IN are operators that check if a value is in a set of values. **IN** returns **true** when the outer query row matches a subquery row. **NOT IN** returns **true** when the outer query row does not match any subquery row.

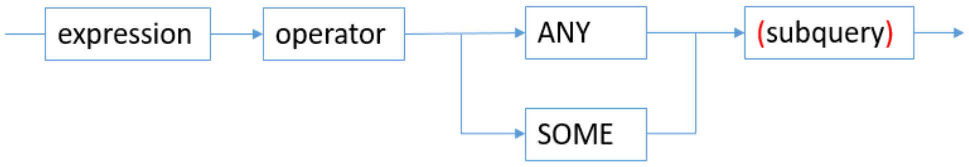
  Syntax:

  ```
  WHERE column_name IN/NOT IN (subquery)
  ```

- ANY/SOME

  ANY indicates that any value in a subquery can match a value in an outer query. **SOME** is the same as **ANY**, but the syntax is different.

  The subquery can return only one column. The expression on the left uses operators (=, <>, <, <=, >, >=) to compare the value with each subquery row. The result must be a Boolean value. 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).

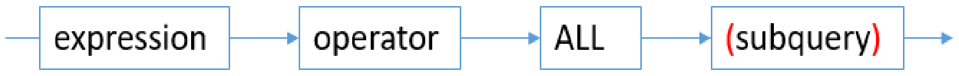
  

  Syntax:

```
WHERE column_name operator ANY/SOME (subquery)
```

- ALL

  The subquery on the right must return only one field. The expression on the left uses operators (=, <>, <, <=, >, >=) to compare the value with each subquery row. The result must be a Boolean value. 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.



Syntax:

```
WHERE column_name operator ALL (subquery)
```

Table 6-1 ALL conditions

Condition	Description
column_name > ALL(...)	The **column_name** value must be greater than the maximum value of a set to be true.
column_name >= ALL(...)	The **column_name** value must be greater than or equal to the maximum value of a set to be true.
column_name < ALL(...)	The **column_name** value must be smaller than the minimum value of a set to be true.
column_name <= ALL(...)	The **column_name** value must be smaller than or equal to the minimum value of a set to be true.
column_name <> ALL(...)	The **column_name** value cannot be equal to any value in a set to be true.
column_name = ALL(...)	The **column_name** value must be equal to any value in a set to be true.

Example

Create the **course** table and insert data into the table.

```
CREATE TABLE course(cid VARCHAR(10) COMMENT 'No.course',cname VARCHAR(10) COMMENT 'course
name',teid VARCHAR(10) COMMENT 'No.teacher');

INSERT INTO course VALUES('01' , 'course1' , '02');
INSERT INTO course VALUES('02' , 'course2' , '01');
INSERT INTO course VALUES('03' , 'course3' , '03');
```

Create the **teacher** table and insert data into the table.

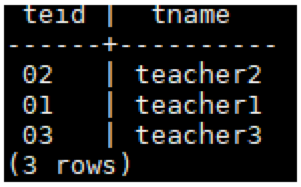
```
CREATE TABLE teacher(teid VARCHAR(10) COMMENT 'Teacher ID',tname VARCHAR(10)
COMMENT'Teacher name');
```

```
INSERT INTO teacher VALUES('01' , 'teacher1');
INSERT INTO teacher VALUES('02' , 'teacher2');
INSERT INTO teacher VALUES('03' , 'teacher3');
INSERT INTO teacher VALUES('04' , 'teacher4');
```

- EXISTS/NOT EXISTS example

Query the teacher records in the course table.

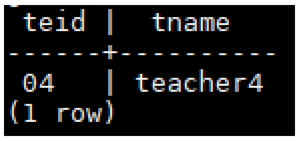
```
SELECT * FROM teacher WHERE EXISTS (SELECT * FROM course WHERE course.teid = teacher.teid);
```



```
 teid |   tname
------+----------
 02   | teacher2
 01   | teacher1
 03   | teacher3
(3 rows)
```

Query the teacher records that are not in the **course** table.

```
SELECT * FROM teacher WHERE NOT EXISTS (SELECT * FROM course WHERE course.teid = teacher.teid);
```

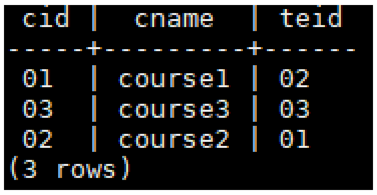


```
 teid |   tname
------+----------
 04   | teacher4
(1 row)
```

- IN/NOT IN example

Query the **course** table for teacher information based on the teacher ID.

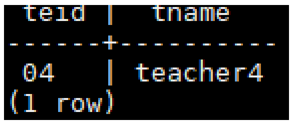
```
SELECT * FROM course WHERE teid IN (SELECT teid FROM teacher );
```



```
 cid |  cname   | teid
-----+---------+------
 01  | course1 | 02
 03  | course3 | 03
 02  | course2 | 01
(3 rows)
```

Query the information about teachers who are not in the **course** table.

```
SELECT * FROM teacher WHERE teid NOT IN (SELECT teid FROM course );
```



```
 teid |   tname
------+----------
 04   | teacher4
(1 row)
```

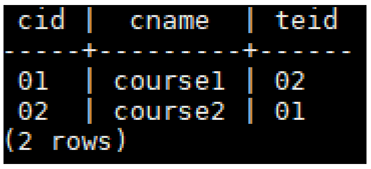
- ANY/SOME example

Compare the main query fields on the left with the subquery fields on the right to obtain the required result set.

```
SELECT * FROM course WHERE teid < ANY (SELECT teid FROM teacher where teid<>'04');
```

or

```
SELECT * FROM course WHERE teid < some (SELECT teid FROM teacher where teid<>'04');
```

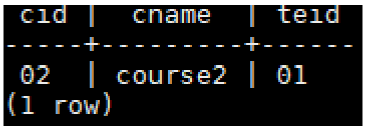


```
 cid |  cname   | teid
-----+---------+------
 01  | course1 | 02
 02  | course2 | 01
(2 rows)
```

- ALL example

The value in the **teid** column must be smaller than the minimum value in the set to be true.

```
SELECT * FROM course WHERE teid < ALL(SELECT teid FROM teacher WHERE teid<>'01');
```



```
 cid |  cname  | teid
-----+---------+------
 02  | course2 | 01
(1 row)
```

Important Notes

- Duplicate subquery statements are not allowed in an SQL statement.
- Avoid scalar sub-queries whenever possible. A scalar subquery is a subquery whose result is one value and whose condition expression uses an equal operator.
- Do not use subqueries in the SELECT target columns. Otherwise, the plan cannot be pushed down, affecting the execution performance.
- It is recommended that the nested subqueries cannot exceed two layers. Subqueries cause temporary table overhead. Therefore, complex queries must be optimized based on service logic.

A subquery can be nested in the SELECT statement to implement a more complex query. A subquery can also use the results of other queries in the WHERE clause to better filter data. However, subqueries may cause query performance problems and make code difficult to read and understand. Therefore, when using SQL subqueries in databases such as GaussDB, use them based on the site requirements.

6.4 GaussDB(DWS) WITH Expressions

The WITH expression is used to define auxiliary statements used in large queries. These auxiliary statements are usually called common table expressions (CTE), which can be understood as a named subquery. The subquery can be referenced multiple times by its name in the quey.

An auxiliary statement may use **SELECT**, **INSERT**, **UPDATE**, or **DELETE**. The **WITH** clause can be attached to a main statement, which can be a **SELECT**, **INSERT**, or **DELETE** statement.

SELECT in WITH

This section describes the usage of **SELECT** in a **WITH** clause.

Syntax

```
[WITH [RECURSIVE] with_query [, ...] ] SELECT ...
```

The syntax of **with_query** is as follows:

```
with_query_name [ ( column_name [, ...] ) ]
   AS [ [ NOT ] MATERIALIZED ] ( {select | values | insert | update | delete} )
```

CAUTION

- If you use **MATERIALIZED**, the subquery runs once and its result set is saved. If you use **NOT MATERIALIZED**, the subquery is replaced with its reference in the main query.
- The SQL statement specified by the AS statement of a CTE must be a statement that can return query results. It can be a common **SELECT** query statement or other data modification statements such as **INSERT**, **UPDATE**, **DELETE**, and **VALUES**. When using a data modification statement, you need to use the **RETURNING** clause to return tuples. Example:
```
WITH s AS (INSERT INTO t VALUES(1) RETURNING a) SELECT * FROM s;
```
- A **WITH** expression indicates the CTE definition in a SQL statement block. Multiple CTEs can be defined at the same time. You can specify column names for each CTE or use the aliases of the columns in the query output. Example:
```
WITH s1(a, b) AS (SELECT x, y FROM t1), s2 AS (SELECT x, y FROM t2) SELECT * FROM s1 JOIN s2 ON
s1.a=s2.x;
```
  This statement defines two CTEs: **s1** and **s2**. **s1** specifies the column names **a** and **b**, and **s2** does not specify the column names. Therefore, the column names are the output column names **x** and **y**.
- Each CTE can be referenced zero, one, or more times in the main query.
- CTEs with the same name cannot exist in the same statement block. If CTEs with the same name exist in different statement blocks, the CTE in the nearest statement block is referenced.
- An SQL statement may contain multiple SQL statement blocks. Each statement block can contain a **WITH** expression. The CTE in each **WITH** expression can be referenced in the current statement block, subsequent CTEs of the current statement block, and sub-layer statement blocks, however, it cannot be referenced in the parent statement block. The definition of each CTE is also a statement block. Therefore, a WITH expression can also be defined in the statement block.

The purpose of SELECT in WITH is to break down complex queries into simple parts. Example:

```
WITH regional_sales AS (
     SELECT region, SUM(amount) AS total_sales
     FROM orders
     GROUP BY region
 ), top_regions AS (
     SELECT region
     FROM regional_sales
     WHERE total_sales > (SELECT SUM(total_sales)/10 FROM regional_sales)
 )
 SELECT region,
        product,
        SUM(quantity) AS product_units,
        SUM(amount) AS product_sales
 FROM orders
 WHERE region IN (SELECT region FROM top_regions)
 GROUP BY region, product;
```

The **WITH** clause defines two auxiliary statements: **regional_sales** and **top_regions**. The output of **regional_sales** is used in **top_regions**, and the output of **top_regions** is used in the main **SELECT** query. This example can be written without **WITH**. In that case, it must be written with a two-layer nested sub-SELECT statement, making the query longer and difficult to maintain.

Recursive WITH Query

By declaring the keyword **RECURSIVE**, a WITH query can reference its own output.

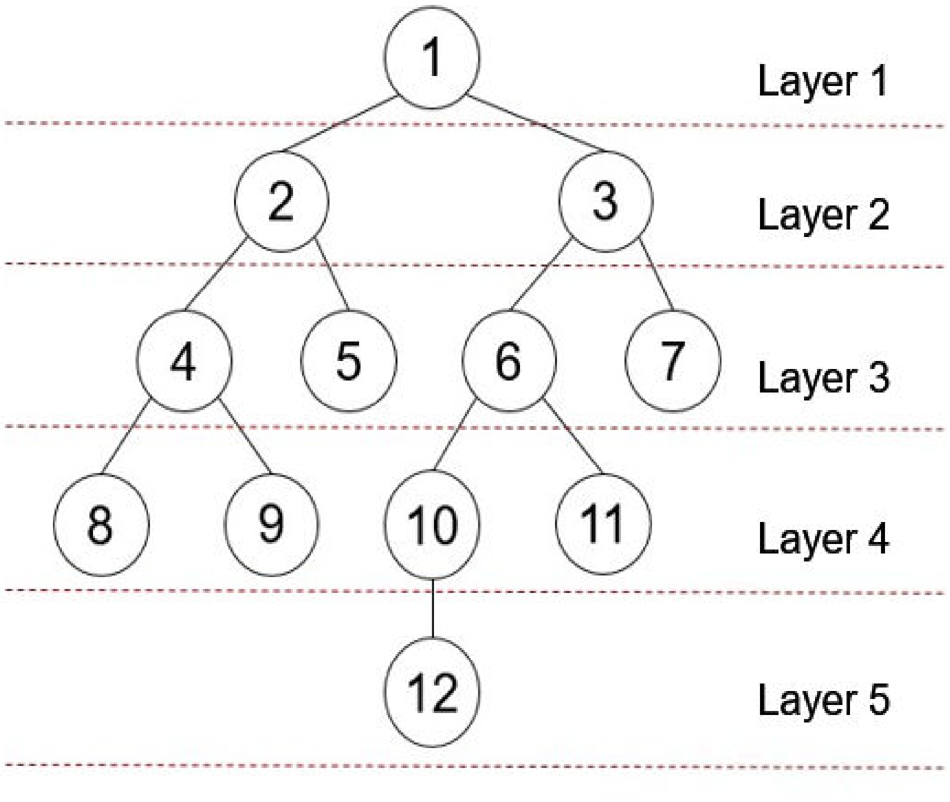
The common form of a recursive WITH query is as follows:

```
non_recursive_term UNION [ALL] recursive_term
```

UNION performs deduplication when merging sets, while **UNION ALLL** directly merges result sets without deduplication. Only recursive items can contain references to the output of the query itself.

When using recursive WITH, ensure that the recursive item of the query does not return a tuple. Otherwise, the query will loop infinitely.

The table **tree** is used to store information about all nodes in the following figure.



The table definition statement is as follows:

```
CREATE TABLE tree(id INT, parentid INT);
```

The data in the table is as follows:

```
INSERT INTO tree VALUES(1,0),(2,1),(3,1),(4,2),(5,2),(6,3),(7,3),(8,4),(9,4),(10,6),(11,6),(12,10);

SELECT * FROM tree;
 id | parentid
----+----------
  1 |        0
  2 |        1
  3 |        1
  4 |        2
  5 |        2
  6 |        3
  7 |        3
  8 |        4
  9 |        4
 10 |        6
 11 |        6
 12 |       10
(12 rows)
```

You can run the following **WITH RECURSIVE** statement to return the nodes and hierarchy information of the entire tree starting from node 1 at the top layer:

```
WITH RECURSIVE nodeset AS
(
-- recursive initializing query
SELECT id, parentid, 1 AS level FROM tree
WHERE id = 1
UNION ALL
-- recursive join query
SELECT tree.id, tree.parentid, level + 1 FROM tree, nodeset
WHERE tree.parentid = nodeset.id
)
SELECT * FROM nodeset ORDER BY id;
```

In the preceding query, a typical **WITH RECURSIVE** expression contains the CTE of at least one recursive query. The CTE is defined as a **UNION ALL** set operation. The first branch is the recursive start query, and the second branch is the recursive join query, the first part is referenced for continuous recursive join. When this statement is executed, the recursive start query is executed once, and the join query is executed several times. The results are added to the start query result set until the results of some join queries are empty.

The command output is as follows:

```
 id | parentid | level
----+----------+-------
  1 |        0 |     1
  2 |        1 |     2
  3 |        1 |     2
  4 |        2 |     3
  5 |        2 |     3
  6 |        3 |     3
  7 |        3 |     3
  8 |        4 |     4
  9 |        4 |     4
 10 |        6 |     4
 11 |        6 |     4
 12 |       10 |     5
(12 rows)
```

According to the returned result, the start query result contains the result set whose level is 1. The join query is executed for five times. The result sets whose levels are 2, 3, 4, and 5 are output for the first four times. During the fifth execution, there is no record whose parentid is the same as the output result set ID, that is, there is no redundant child node. Therefore, the query ends.

NOTE

GaussDB(DWS) supports distributed execution of **WITH RECURSIVE** expressions. **WITH RECURSIVE** involves cyclic calculation. Therefore, GaussDB(DWS) introduces the **max_recursive_times** parameter to control the maximum number of cycles of WITH RECURSIVE. The default value is **200**. If the number of cycles exceeds **200**, an error is reported.

Data Modification Statements in WITH

Use the **INSERT**, **UPDATE**, and **DELETE** commands in the WITH clause. This allows the user to perform multiple different operations in the same query. The following is an example:

```
WITH moved_tree AS (
     DELETE FROM tree
     WHERE parentid = 4
     RETURNING * )
 INSERT INTO tree_log
 SELECT * FROM moved_tree;
```

The preceding query example actually moves rows from **tree** to **tree_log**. The **DELETE** command in the **WITH** clause deletes the specified rows from **tree**, returns their contents through the **RETURNING** clause, and then the main query reads the output and inserts it into **tree_log**.

To retrieve the modified content instead of the target table, the data modification statement in the **WITH** clause should include the **RETURNING** clause. This clause creates a temporary table that can be accessed by the rest of the query. If a data modification statement in the **WITH** statement lacks a **RETURNING** clause, it cannot form a temporary table and cannot be referenced in the remaining queries.

If the **RECURSIVE** keyword is declare, recursive self-reference is not allowed in data modification statements. In some cases, you can bypass this restriction by referencing the output of recursive the **WITH** statement. For example:

```
WITH RECURSIVE included_parts(sub_part, part) AS (
     SELECT sub_part, part FROM parts WHERE part = 'our_product'
   UNION ALL
     SELECT p.sub_part, p.part
     FROM included_parts pr, parts p
     WHERE p.part = pr.sub_part
   )
DELETE FROM parts
   WHERE part IN (SELECT part FROM included_parts);
```

This query will remove all direct or indirect subparts of a product.

The substatements in the **WITH** clause are executed at the same time as the main query. Therefore, when using the data modification statement in a WITH statement, the actual update order is in an unpredictable manner. All statements are executed in the same snapshot, and the effect of the statements is invisible on the target table. This mitigates the unpredictability of the actual order of row updates and means that **RETURNING** data is the only way to convey changes between different **WITH** substatements and the main query.

In this example, the outer layer **SELECT** can return the data before the update.

```
WITH t AS (
     UPDATE tree SET id = id + 1
     RETURNING * )
SELECT * FROM tree;
```

In this example, the external SELECT returns the updated data.

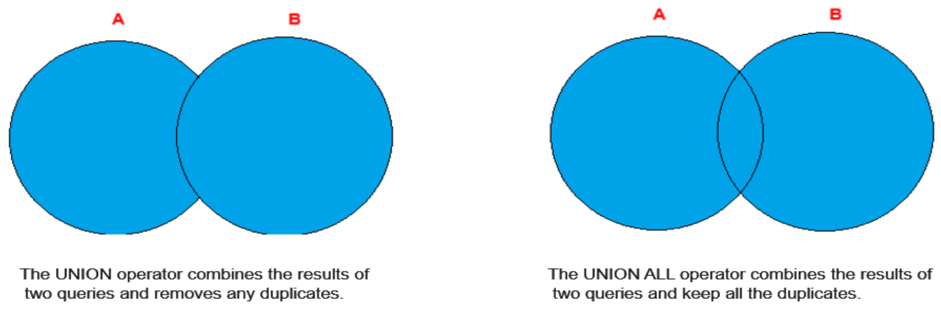
```
WITH t AS (
UPDATE tree SET id = id + 1
     RETURNING * )
SELECT * FROM t;
```

The same row cannot be updated twice in a single statement. Otherwise, the update effect will be unpredictable. If only one update takes effect, it is difficult (and sometimes impossible) to predict which one takes effect.

6.5 Usage of GaussDB(DWS) UNION

UNION is a powerful SQL operator that combines the result sets of two or more SELECT statements into one. During combination, the number of columns and data types in the two tables must be the same and correspond to each other. Use the **UNION** or **UNION** ALL keyword between SELECT statements.

UNION removes duplicate rows, while UNION ALL keeps them. Deduplication is time-consuming, so UNION ALL can be faster than UNION if the data sets are already distinct by logic.


A
B
The UNION operator combines the results of two queries and removes any duplicates.
A
B
The UNION ALL operator combines the results of two queries and keep all the duplicates.

Syntax

```
SELECT column,... FROM table1 UNION [ALL]SELECT column,... FROM table2
```

Example

Step 1 Create the student information table **student** (ID, name, gender, and school).

```
SET current_schema=public;
DROP TABLE IF EXISTS student;
CREATE table student(
sId VARCHAR(10) NOT NULL,
sname VARCHAR(10) NOT NULL,
sgender VARCHAR(10) NOT NULL,
sschool VARCHAR(10) NOT NULL);
```

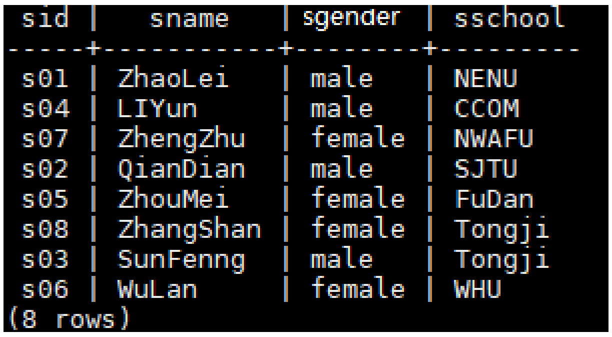
Step 2 Insert data into the **student** table.

```
INSERT INTO student VALUES('s01' , 'ZhaoLei' , 'male', 'NENU');
INSERT INTO student VALUES('s02' , 'QianDian' , 'male', 'SJTU');
INSERT INTO student VALUES('s03' , 'SunFenng' , 'male', 'Tongji');
INSERT INTO student VALUES('s04' , 'LIYun' , 'male', 'CCOM');
INSERT INTO student VALUES('s05' , 'ZhouMei' , 'female', 'FuDan');
INSERT INTO student VALUES('s06' , 'WuLan' , 'female', 'WHU');
INSERT INTO student VALUES('s07' , 'ZhengZhu' , 'female', 'NWAFU');
INSERT INTO student VALUES('s08' , 'ZhangShan' , 'female', 'Tongji');
```

Step 3 View the student table.

```
SELECT * FROM student;
```

Information similar to the following is displayed.



```
 sid |    sname    | sgender | sschool
-----+------------+--------+---------
 s01 | ZhaoLei    | male    | NENU
 s04 | LIYun      | male    | CCOM
 s07 | ZhengZhu   | female  | NWAFU
 s02 | QianDian   | male    | SJTU
 s05 | ZhouMei    | female  | FuDan
 s08 | ZhangShan  | female  | Tongji
 s03 | SunFenng   | male    | Tongji
 s06 | WuLan      | female  | WHU
(8 rows)
```

Step 4 Create the teacher information table **teacher** (ID, name, gender, and school).

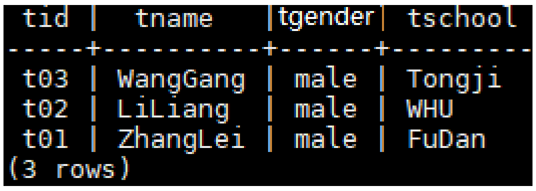
```
DROP TABLE IF EXISTS teacher;
CREATE table teacher(
tid VARCHAR(10) NOT NULL,
tname VARCHAR(10) NOT NULL,
tgender VARCHAR(10) NOT NULL,
tschool VARCHAR(10) NOT NULL);
```

Step 5 Insert data to the **teacher** table.

```
INSERT INTO teacher VALUES('t01' , 'ZhangLei', 'male', 'FuDan');
INSERT INTO teacher VALUES('t02' , 'LiLiang', 'male', 'WHU');
INSERT INTO teacher VALUES('t03' , 'WangGang', 'male', 'Tongji');
```

Step 6 Query the **teacher** table.

```
SELECT * FROM teacher;
```

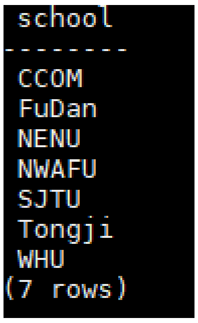


```
 tid |   tname    |tgender| tschool
-----+-----------+------+---------
 t03 | WangGang | male | Tongji
 t02 | LiLiang  | male | WHU
 t01 | ZhangLei | male | FuDan
(3 rows)
```

Step 7 Use **UNION** (combine and deduplicate) to obtain the schools of students and teachers and sort the schools in ascending order by initial letter of the school name.

```
SELECT t.school  FROM (
    SELECT sschool AS school
     FROM student
     UNION
     SELECT tschool AS school
     FROM teacher
 ) t
 ORDER BY t.school ASC;
```

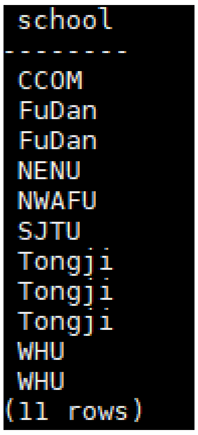
Information similar to the following is displayed.



```
 school
--------
 CCOM
 FuDan
 NENU
 NWAFU
 SJTU
 Tongji
 WHU
(7 rows)
```

Step 8 Use **UNION ALL** (combine without deduplication) to obtain the schools of all students and teachers and sort the schools by initial letter of the school name in ascending order.

```
SELECT t.school  FROM (
    SELECT sschool AS school
     FROM student
     UNION ALL
     SELECT tschool AS school
     FROM teacher
  ) t
  ORDER BY t.school ASC;
```



```
 school
--------
 CCOM
 FuDan
 FuDan
 NENU
 NWAFU
 SJTU
 Tongji
 Tongji
 Tongji
 WHU
 WHU
(11 rows)
```

Step 9 Use **UNION ALL** (combine the result sets of SQL statements with **WHERE** clause) to get all information about students and teachers from ''Tongji' and sort by student and teacher number in ascending order.

```
SELECT t.*  FROM  (
  SELECT Sid AS id,Sname AS name,Sgender AS gender,Sschool AS school
  FROM student
  WHERE Sschool='Tongji'
  UNION ALL
  SELECT Tid AS id,Tname AS name,Tgender AS gender,Tschool AS school
  FROM teacher
  WHERE Tschool='Tongji'
) t
  ORDER BY t.id ASC;
```

----End

Summary

In actual service scenarios, pay attention to the following points when using **UNION** and **UNION ALL**:

- The number of SQL fields and field types on the left and right sides must be the same.
- Check whether data deduplication (deduplication before combination or during combination) is needed based on service requirements.
- Based on the data volume, valuate the SQL execution efficiency and determine whether to use temporary tables.
- Select **UNION** or **UNION ALL** wisely and consider the complexity when writing SQL statements.

7 GaussDB(DWS) Sorting Rules

The collation feature allows specifying the data sorting order and data classification rules in a character set. This alleviates the restriction that the **LC_COLLATE** and **LC_CTYPE** settings of a database cannot be changed after its creation.

Overview

Every expression of a collatable data type has a collation. (The built-in collatable data types are text, varchar, and char. User-defined base types can also be marked collatable, and of course a domain over a collatable data type is collatable.) If the expression is a column reference, the collation of the expression is the defined collation of the column. If the expression is a constant, the collation is the default collation of the data type of the constant. The collation of a more complex expression is derived from the collations of its inputs.

Collation Combination Principles

- The collation of an expression can be the default collation, which means the locale settings defined for the database. It is also possible for an expression's collation to be indeterminate. In such cases, ordering operations and other operations that need to know the collation will fail.
- For a function or operator call, the collation that is derived by examining the argument collations is used at run time for performing the specified operation. If the result of the function or operator call is of a collatable data type, the collation is also used as the defined collation of the function or operator expression, in case there is a surrounding expression that requires knowledge of its collation.
- The collation derivation of an expression can be implicit or explicit. This distinction affects how collations are combined when multiple different collations appear in an expression. An explicit collation derivation occurs when a **COLLATE** clause is used; all other collation derivations are implicit. When multiple collations need to be combined, the following rules are used:
  - If any input expression has an explicit collation derivation, then all explicitly derived collations among the input expressions must be the same, otherwise an error is raised. If any explicitly derived collation is present, that is the result of the collation combination.

- Otherwise, all input expressions must have the same implicit collation derivation or the default collation. If any non-default collation is present, that is the result of the collation combination. Otherwise, the result is the default collation.
- If there are conflicting non-default implicit collations among the input expressions, then the combination is deemed to have indeterminate collation. This is not an error condition unless the particular function being invoked requires knowledge of the collation it should apply. If it does, an error will be raised at run-time.

- In a CASE expression, the comparison rule is subject to the COLLATE setting in the WHEN clause.
- Explicit COLLATE derivation takes effect only in the current query (CTE or SUBQUERY). Outside the query, implicit derivation takes effect.

Collation Tips

- Do not use multiple collations in the same query statement. Otherwise, exceptional result sets may be generated.
- Do not use multiple COLLATE clauses to specify a collation.

Case-insensitive Collation Support

Since cluster 8.1.3, GaussDB(DWS) has added the built-in case_insensitive collation, which is case-insensitive to character types in some actions (such as sorting, comparison, and hash).

Constraints:

- Supported character types: char, character, nchar, and varchar/character varying/varchar2/nvarchar2/clob/text.
- The character types **char** and **name** are not supported.
- The following encoding formats are not supported: PG_EUC_JIS_2004, PG_MULE_INTERNAL, PG_LATIN10 and PG_WIN874.
- It cannot be specified to **LC_COLLATE** when **CREATE DATABASE** is executed.
- Regular expressions are not supported.
- Record comparison of the character type (for example, **record_eq**) is not supported.
- Time series tables are not supported.
- Skew optimization is not supported.
- RoughCheck optimization is not supported.

Examples

The COLLATE clause is specified in the statement.

```
SELECT 'a' = 'A', 'a' = 'A' COLLATE case_insensitive;
 ?column? | ?column?
----------+----------
 f        | t
(1 row)
```

Set the column attribute to **case_insensitive** when creating a table.

```
CREATE TABLE t1 (a text collate case_insensitive);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using round-robin as the distribution mode by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
\d t1
           Table "public.t1"
 Column | Type |        Modifiers
--------+------+--------------------------
 a      | text | collate case_insensitive

INSERT INTO t1 values('a'),('A'),('b'),('B');
INSERT 0 4
```

This parameter is specified during table creation and does not need to be specified during query.

```
SELECT a, a='a' FROM t1;
 a | ?column?
---+----------
 A | t
 B | f
 a | t
 b | f
(4 rows)
SELECT a, count(1) FROM t1 GROUP BY a;
 a | count
---+-------
 a |     2
 B |     2
(2 rows)
```

CASE expression, which is subject to the COLLATE setting in the WHEN clause.

```
SELECT a,case a when 'a' collate case_insensitive then 'case1' when 'b' collate "C" then 'case2' else 'case3'
end FROM t1;
 a | case
---+-------
 A | case1
 B | case3
 a | case1
 b | case2
(4 rows)
```

Implicit derivation across subqueries.

```
SELECT * FROM (SELECT a collate "C" from t1) WHERE a in ('a','b');
 a
---
 a
 b
(2 rows)
SELECT * FROM t1,(SELECT a collate "C" from t1) t2 WHERE t1.a=t2.a;
ERROR:  could not determine which collation to use for string hashing
HINT:  Use the COLLATE clause to set the collation explicitly.
```

CAUTION

- **collate case_insensitive** is an insensitive sorting, and the result set is uncertain. If sensitive sorting is used after **collate case_insensitive** sorting, the result set may be unstable. Therefore, do not use sensitive sorting and insensitive sorting together in statements.
- If **collate case_insensitive** is used to specify character behaviors as case-insensitive, the performance will be affected. If you require high performance, exercise caution when configuring this parameter.

8 GaussDB(DWS) User-Defined Functions

NOTE

The hybrid data warehouse (deployed in standalone mode) does not support user-defined functions.

8.1 GaussDB(DWS) PL/Java Functions

With the GaussDB(DWS) PL/Java functions, you can choose your favorite Java IDE to write Java methods and install the JAR files containing these methods into the GaussDB(DWS) database before invoking them. GaussDB(DWS) PL/Java is developed based on open-source PL/Java 1.5.5 and uses JRE 1.8.0_322.

Constraints

Java UDF can be used for some Java logical computing. You are not advised to encapsulate services in Java UDF.

- You are not advised to connect to a database in any way (for example, JDBC) in Java functions.
- Currently, only data types listed in **Table 8-1** are supported. Other data types, such as user-defined data types and complex data types (for example, Java array and its derived types) are not supported.
- Currently, UDAF and UDTF are not supported.

Examples

Before using PL/Java, you need to pack the implementation of Java methods into a JAR package and deploy it into the database. Then, create functions as a database administrator. For compatibility purposes, use JRE 1.8.0_322 for compilation.

Step 1 Compile a JAR package.

Java method implementation and JAR package archiving can be achieved in an integrated development environment (IDE). The following is a simple example of compilation and archiving through command lines. You can create a JAR package that contains a single method in the similar way.

First, prepare an **Example.java** file that contains a method for converting substrings to uppercase. In the following example, **Example** is the class name and **upperString** is the method name:

```
public class Example
{
    public static String upperString (String text, int beginIndex, int endIndex)
    {
        return text.substring(beginIndex, endIndex).toUpperCase();
    }
}
```

Then, create a **manifest.txt** file containing the following content:

```
Manifest-Version: 1.0
Main-Class: Example
Specification-Title: "Example"
Specification-Version: "1.0"
Created-By: 1.6.0_35-b10-428-11M3811
Build-Date: 08/14/2018 10:09 AM
```

Manifest-Version specifies the version of the **manifest** file. **Main-Class** specifies the main class used by the .jar file. **Specification-Title** and **Specification-Version** are the extended attributes of the package. **Specification-Title** specifies the title of the extended specification and **Specification-Version** specifies the version of the extended specification. **Created-By** specifies the person who created the file. **Build-Date** specifies the date when the file was created.

Finally, archive the .java file and package it into **javaudf-example.jar**.

```
javac Example.java
jar cfm javaudf-example.jar manifest.txt Example.class
```

NOTICE

JAR package names must comply with JDK rules. If a name contains invalid characters, an error occurs when a function is deployed or used.

Step 2 Deploy the JAR package.

Place the JAR package on the OBS server using the method described in For details, see "Uploading a File" in *Object Storage Service Console Operation Guide*.. Then, create the AK/SK. For details about how to obtain the AK/SK, see section **Creating Access Keys (AK and SK)**. Log in to the database and run the **gs_extend_library** function to import the file to GaussDB(DWS).

```
SELECT gs_extend_library('addjar', 'obs://bucket/path/javaudf-example.jar 
accesskey=access_key_value_to_be_replaced  secretkey=secret_access_key_value_to_be_replaced 
region=region_name libraryname=example');
```

For details about how to use the **gs_extend_library** function, see **Manage JAR packages and files**. Change the values of AK and SK as needed. Replace *region_name* with an actual region name.

Step 3 Use a PL/Java function.

Log in to the database as a user who has the **sysadmin** permission (for example, dbadmin) and create the **java_upperstring** function:

```
CREATE FUNCTION java_upperstring(VARCHAR, INTEGER, INTEGER)
    RETURNS VARCHAR
```

```
  AS 'Example.upperString'
LANGUAGE JAVA;
```

NOTE

- The data type defined in the java_upperstring function should be a type in GaussDB(DWS) and match the data type defined in **Step 1** in the upperString method in Java. For details about the mapping between GaussDB(DWS) and Java data types, see **Table 8-1**.
- The AS clause specifies the class name and static method name of the Java method invoked by the function. The format is *Class name*.*Method name*. The class name and method name must match the Java class and method defined in **Step 1**.
- To use PL/Java functions, set **LANGUAGE** to **JAVA**.
- For details about CREATE FUNCTION, see **Create functions**.

Execute the java_upperstring function.

```
SELECT java_upperstring('test', 0, 1);
```

The expected result is as follows:

```
 java_upperstring
---------------------
 T
(1 row)
```

Step 4 Authorize a common user to use the PL/Java function.

Create a common user named **udf_user**.

```
CREATE USER udf_user PASSWORD 'password';
```

This command grants user **udf_user** the permission for the java_upperstring function. Note that the user can use this function only if it also has the permission for using the schema of the function.

```
GRANT ALL PRIVILEGES ON SCHEMA public TO udf_user;
GRANT ALL PRIVILEGES ON FUNCTION java_upperstring(VARCHAR, INTEGER, INTEGER) TO udf_user;
```

Log in to the database as user **udf_user**.

```
SET SESSION SESSION AUTHORIZATION udf_user PASSWORD 'password';
```

Execute the java_upperstring function.

```
SELECT public.java_upperstring('test', 0, 1);
```

The expected result is as follows:

```
 java_upperstring
---------------------
 T
(1 row)
```

Step 5 Delete the function.

If you no longer need this function, delete it.

```
DROP FUNCTION java_upperstring;
```

Step 6 Uninstall the JAR package.

Use the gs_extend_library function to uninstall the JAR package.

```
SELECT gs_extend_library('rmjar', 'libraryname=example');
```

----End

SQL Definition and Usage

- **Manage JAR packages and files.**

  A database user having the **sysadmin** permission can use the gs_extend_library function to deploy, view, and delete JAR packages in the database. The syntax of the function is as follows:

  ```
  SELECT gs_extend_library('[action]', '[operation]');
  ```

  NOTE

  - **action**: operation action. The options are as follows:
    - **ls**: Displays JAR packages in the database and checks the MD5 value consistency of files on each node.
    - **addjar**: deploys a JAR package on the OBS server in the database.
    - **rmjar**: Deletes JAR packages from the database.
  - **operation**: operation string. The format can be either of the following:

    obs://[bucket]/[source_filepath] accesskey=[accesskey] secretkey=[secretkey] region=[region] libraryname=[libraryname]
    - **bucket**: name of the bucket to which the OBS file belongs. It is mandatory.
    - **source_filepath**: file path on the OBS server. Only .jar files are supported.
    - **accesskey**: key obtained for accessing the OBS service. It is mandatory.
    - **secret_key**: secret key obtained for the OBS service. It is mandatory.
    - **region**: region where the OBS bucket stored in the JAR package of a user-defined function belongs to. This parameter is mandatory.
    - **libraryname**: user-defined library name, which is used to invoke JAR files in GaussDB(DWS). If **action** is set to **addjar** or **rmjar**, **libraryname** must be specified. If **action** is set to **ls**, **libraryname** is optional. Note that a user-defined library name cannot contain the following characters: /|;&$<>\'{}"()[]~*?!

- Create functions.

  PL/Java functions can be created using the **CREATE FUNCTION** syntax and are defined as **LANGUAGE JAVA**, including the **RETURNS** and **AS** clauses.

  – To use **CREATE FUNCTION**, specify the name and parameter type for the function to be created.

  – The **RETURNS** clause specifies the return type for the function.

  – The **AS** clause specifies the class name and static method name of the Java method to be invoked. If the **NULL** value needs to be transferred to the Java method as an input parameter, specify the name of the Java encapsulation class corresponding to the parameter type. For details, see **NULL Handling**.

  – For details about the syntax, see CREATE FUNCTION.

  ```
  CREATE [ OR REPLACE ] FUNCTION function_name
  ( [ { argname [ argmode ] argtype [ { DEFAULT | := | = } expression ]} [, ...] ])
  [ RETURNS rettype [ DETERMINISTIC ] ]
  LANGAUGE JAVA
  [
     { IMMUTABLE | STABLE | VOLATILE }
     | [ NOT ] LEAKPROOF
     | WINDOW
     | { CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT |STRICT }
     | {[ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER | AUTHID DEFINER |
  AUTHID CURRENT_USER}
     | { FENCED }
     | COST execution_cost
     | ROWS result_rows
  ```

```
    | SET configuration_parameter { {TO |=} value | FROM CURRENT}
] [...]
{
    AS 'class_name.method_name' ( { argtype } [, ...] )
}
```

- Use functions.

  During execution, PL/Java searches for the Java class specified by a function among all the deployed JAR packages, which are ranked by name in alphabetical order, invokes the Java method in the first found class, and returns results.

- Delete functions.

  PL/Java functions can be deleted by using the **DROP FUNCTION** syntax. For details about the syntax, see DROP FUNCTION.

  ```
  DROP FUNCTION [ IF EXISTS ] function_name [ ( [ {[ argmode ] [ argname ] argtype} [, ...] ] )
  [ CASCADE | RESTRICT ] ];
  ```

  To delete an overloaded function (for details, see **Overloaded Functions**), specify **argtype** in the function. To delete other functions, simply specify **function_name**.

- Authorize permissions for functions.

  Only user **sysadmin** can create PL/Java functions. It can also grant other users the permission to use the PL/Java functions. For details about the syntax, see GRANT.

  ```
  GRANT { EXECUTE | 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 ];
  ```

Mapping for Basic Data Types

Table 8-1 PL/Java mapping for default data types

GaussDB(DWS)	Java
BOOLEAN	boolean
"char"	byte
bytea	byte[]
SMALLINT	short
INTEGER	int
BIGINT	long
FLOAT4	float
FLOAT8	double
CHAR	java.lang.String
VARCHAR	java.lang.String
TEXT	java.lang.String

GaussDB(DWS)	Java
name	java.lang.String
DATE	java.sql.Timestamp
TIME	java.sql.Time (stored value treated as local time)
TIMETZ	java.sql.Time
TIMESTAMP	java.sql.Timestamp
TIMESTAMPTZ	java.sql.Timestamp

Array Type Processing

GaussDB(DWS) can convert basic array types. You only need to append a pair of square brackets ([]) to the data type when creating a function.

```
CREATE FUNCTION java_arrayLength(INTEGER[])
    RETURNS INTEGER
    AS 'Example.getArrayLength'
LANGUAGE JAVA;
```

Java code is similar to the following:

```
public class Example
{
    public static int getArrayLength(Integer[] intArray)
    {
        return intArray.length;
    }
}
```

Invoke the following statement:

```
SELECT java_arrayLength(ARRAY[1, 2, 3]);
```

The expected result is as follows:

```
java_arrayLength
---------------------
3
(1 row)
```

NULL Handling

NULL values cannot be handled for GaussDB(DWS) data types that are mapped and can be converted to simple Java types by default. If you use a Java function to obtain and process the **NULL** value transferred from GaussDB(DWS), specify the Java encapsulation class in the **AS** clause as follows:

```
CREATE FUNCTION java_countnulls(INTEGER[])
    RETURNS INTEGER
    AS 'Example.countNulls(java.lang.Integer[])'
LANGUAGE JAVA;
```

Java code is similar to the following:

```
public class Example
{
    public static int countNulls(Integer[] intArray)
    {
        int nullCount = 0;
        for (int idx = 0; idx < intArray.length; ++idx)
        {
            if (intArray[idx] == null)
            nullCount++;
        }
        return nullCount;
    }
}
```

Invoke the following statement:

```
SELECT java_countNulls(ARRAY[null, 1, null, 2, null]);
```

The expected result is as follows:

```
java_countNulls
--------------------
3
(1 row)
```

Overloaded Functions

PL/Java supports overloaded functions. You can create functions with the same name or invoke overloaded functions from Java code. The procedure is as follows:

Step 1 Create overloaded functions.

For example, create two Java methods with the same name, and specify the methods dummy(int) and dummy(String) with different parameter types.

```
public class Example
{
    public static int dummy(int value)
    {
        return value*2;
    }
    public static String dummy(String value)
    {
        return value;
    }
}
```

In addition, create two functions with the same names as the above two functions in GaussDB(DWS).

```
CREATE FUNCTION java_dummy(INTEGER)
    RETURNS INTEGER
    AS 'Example.dummy'
LANGUAGE JAVA;

CREATE FUNCTION java_dummy(VARCHAR)
    RETURNS VARCHAR
    AS 'Example.dummy'
LANGUAGE JAVA;
```

Step 2 Invoke the overloaded functions.

GaussDB(DWS) invokes the functions that match the specified parameter type. The results of invoking the above two functions are as follows:

```
SELECT java_dummy(5);
 java_dummy
-----------------
          10
```

```
(1 row)

SELECT java_dummy('5');
 java_dummy
---------------
5
(1 row)
```

Note that GaussDB(DWS) may implicitly convert data types. Therefore, you are advised to specify the parameter type when invoking an overloaded function.

```
SELECT java_dummy(5::varchar);
 java_dummy
----------------
5
(1 row)
```

In this case, the specified parameter type is preferentially used for matching. If there is no Java method matching the specified parameter type, the system implicitly converts the parameter and searches for Java methods based on the conversion result.

```
SELECT java_dummy(5::INTEGER);
 java_dummy
-----------------
10
(1 row)

DROP FUNCTION java_dummy(INTEGER);

SELECT java_dummy(5::INTEGER);
 java_dummy
----------------
5
(1 row)
```

NOTICE

Data types supporting implicit conversion are as follows:

- **SMALLINT**: It can be converted to the **INTEGER** type by default.
- **SMALLINT** and **INTEGER**: They can be converted to the **BIGINT** type by default.
- **TINYINT**, **SMALLINT**, **INTEGER**, and **BIGINT**: They can be converted to the **BOOL** type by default.
- The following data types can be converted to TEXT by default: CHAR, NAME, BIGINT, INTEGER, SMALLINT, TINYINT, RAW, FLOAT4, FLOAT8, BPCHAR, VARCHAR, NVARCHAR2, DATE, TIMESTAMP, TIMESTAMPTZ, NUMERIC, and SMALLDATETIME.
- The following data types can be converted to VARCHAR by default: TEXT, CHAR, BIGINT, INTEGER, SMALLINT, TINYINT, RAW, FLOAT4, FLOAT8, BPCHAR, DATE, NVARCHAR2, TIMESTAMP, NUMERIC, and SMALLDATETIME.

Step 3 Delete the overloaded functions.

To delete an overloaded function, specify the parameter type for the function. Otherwise, the function cannot be deleted.

```
DROP FUNCTION java_dummy(INTEGER);
```

----End

GUC Parameters

- **udf_memory_limit**

  A system-level GUC parameter. It is used to limit the physical memory used by each CN or DN for executing UDFs. The default value is **0.05 * max_process_memory**. You can use the **postgresql.conf** file to modify the parameter setting. The modification takes effect only after the database is restarted.

  NOTICE

  - **udf_memory_limit** is a part of **max_process_memory**. When a CN or DN is started, memory calculated by **udf_memory_limit** minus **200 MB** will be reserved for Worker processes. CN and DN processes are different from the UDF Worker process, and the CN and DN processes will save memory for the UDF Worker process.

    For example, if **max_process_memory** is set to **10GB** on a DN and **udf_memory_limit** is set to **4GB**, the DN can use a maximum of 6.2 GB memory, that is, 10 GB – (4 GB – 200 MB). This case applies even if no UDF is executed. By default, the value of **udf_memory_limit** is **0.05 * max_process_memory**. Querying the **pv_total_memory_detail** view will prove that the value of **process_used_memory** would never exceed the calculation result of **max_process_memory** – (**udf_memory_limit** – **200MB**).
  - Executing a simplest Java UDF on a CN consumes about 50 MB physical memory. You can set this parameter based on the memory usage and concurrency of Java functions to be used. After this parameter is added, you are not advised to set **UDFWorkerMemHardLimit** and **FencedUDFMemoryLimit**.
  - If the parallelism of the UDF process is excessively high and the memory usage exceeds the **udf_memory_limit** value, unexpected situations such as process exit may occur. In this scenario, the execution result may be unreliable. You are advised to set this parameter to reserve sufficient memory based on the site requirements. If the system has the **/var/log/messages** log, check the log to see whether the memory is insufficient because the cgroup memory limit has been reached. If the memory is severely insufficient, the UDF master process may exit. You can view the UDF log for analysis. The default UDF log path is **$GAUSSLOG/cm/cm_agent/pg_log**. For example, if the following log is displayed, the memory resources are insufficient and the UDF master process exits. In this case, you need to check the **udf_memory_limit** parameter.

    0 [BACKEND] FATAL: poll() failed: Bad address, please check the parameter:udf_memory_limit to make sure there is enough memory.

- **FencedUDFMemoryLimit**

  A session-level GUC parameter. It is used to specify the maximum virtual memory used by a single Fenced UDF Worker process initiated by a session.

```
SET FencedUDFMemoryLimit='512MB';
```

The value range of this parameter is (**150 MB**, **1G**). If the value is greater than **1G**, an error will be reported immediately. If the value is less than or equal to **150 MB**, an error will be reported during function invoking.

NOTICE

- If **FencedUDFMemoryLimit** is set to **0**, the virtual memory for a Fenced UDF Worker process will not be limited.
- You are advised to use **udf_memory_limit** to control the physical memory used by Fenced UDF Worker processes. You are not advised to use **FencedUDFMemoryLimit**, especially when Java UDFs are used. If you are clear about the impact of this parameter, set it based on the following information:
  - After a C Fenced UDF Worker process is started, it will occupy about 200 MB virtual memory, and about 16 MB physical memory.
  - After a Java Fenced UDF Worker process is started, it will occupy about 2.5 GB virtual memory, and about 50 MB physical memory.

Exception Handling

If there is an exception in a JVM, PL/Java will export JVM stack information during the exception to a client.

Logging

PL/Java uses the standard Java Logger. Therefore, you can record logs as follows:

```
Logger.getAnonymousLogger().config( "Time is " + new
Date(System.currentTimeMillis()));
```

An initialized Java Logger class is set to the **CONFIG** level by default, corresponding to the **LOG** level in GaussDB(DWS). In this case, log messages generated by Java Logger are all redirected to the GaussDB(DWS) backend. Then, the log messages are written into server logs or displayed on the user interface. MPPDB server logs record information at the **LOG**, **WARNING**, and **ERROR** levels. The SQL user interface displays logs at the **WARNING** and **ERROR** levels. The following table lists mapping between Java Logger levels and GaussDB(DWS) log levels.

Table 8-2 PL/Java log levels

java.util.logging.Level	GaussDB(DWS) Log Level
SERVER	ERROR
WARNING	WARNING
CONFIG	LOG
INFO	INFO

java.util.logging.Level	GaussDB(DWS) Log Level
FINE	DEBUG1
FINER	DEBUG2
FINEST	DEBUG3

You can change Java Logger levels. For example, if the Java Logger level is changed to **SEVERE** by the following Java code, log messages (**msg**) will not be recorded in GaussDB(DWS) logs during **WARNING** logging.

```
Logger log = Logger.getAnonymousLogger();
Log.setLevel(Level.SEVERE);
log.log(Level.WARNING, msg);
```

Security Issues

In GaussDB(DWS), PL/Java is an untrusted language. Only user **sysadmin** can create PL/Java functions. The user can grant other users the permission for using the PL/Java functions. For details, see **Authorize permissions for functions**.

In addition, PL/Java controls user access to file systems, forbidding users from reading most system files, or writing, deleting, or executing any system files in Java methods.

8.2 GaussDB(DWS) PL/pgSQL Functions

PL/pgSQL is similar to PL/SQL of Oracle. It is a loadable procedural language.

The functions created using PL/pgSQL can be used in any place where you can use built-in functions. For example, you can create calculation functions with complex conditions and use them to define operators or use them for index expressions.

SQL is used by most databases as a query language. It is portable and easy to learn. Each SQL statement must be executed independently by a database server.

In this case, when a client application sends a query to the server, it must wait for it to be processed, receive and process the results, and then perform some calculation before sending more queries to the server. If the client and server are not on the same machine, all these operations will cause inter-process communication and increase network loads.

PL/pgSQL enables a whole computing part and a series of queries to be grouped inside a database server. This makes procedural language available and SQL easier to use. In addition, the client/server communication cost is reduced.

- Extra round-trip communication between clients and servers is eliminated.
- Intermediate results that are not required by clients do not need to be sorted or transmitted between the clients and servers.
- Parsing can be skipped in multiple rounds of queries.

PL/pgSQL can use all data types, operators, and functions in SQL.

For details about the PL/pgSQL syntax for creating functions, see **CREATE FUNCTION**. As mentioned earlier, PL/pgSQL is similar to PL/SQL of Oracle and is a loadable procedural language. Its application method is similar to that of **GaussDB(DWS) Stored Procedure**. There is only one difference. Stored procedures have no return values but the functions have.

9 GaussDB(DWS) Stored Procedure

9.1 Overview

What Is a GaussDB(DWS) Stored Procedure?

In GaussDB(DWS), business rules and logics are saved as stored procedures.

A stored procedure is a combination of SQL, PL/SQL, and Java statements. Stored procedures can move the code that executes business rules from applications to databases. In this way, code can be used by multiple programs at a time.

For details about how to create and call a stored procedure, see **CREATE PROCEDURE**.

The functions created using the PL/pgSQL language mentioned in **GaussDB(DWS) PL/pgSQL Functions** are similar to the application methods of stored procedures. Unless otherwise specified, the following sections apply to stored procedures and PL/pgSQL functions.

GaussDB(DWS) Stored Procedure Data Types

A data type refers to a value set and an operation set defined on the value set. A GaussDB(DWS) database consists of tables, each of which is defined by its own columns. Each column corresponds to a data type. GaussDB(DWS) uses corresponding functions to perform operations on data based on data types. For example, GaussDB(DWS) can perform addition, subtraction, multiplication, and division operations on data of numeric values.

9.2 Converting Data Types in GaussDB(DWS) Stored Procedures

Certain data types in the database support implicit data type conversions, such as assignments and parameters invoked by functions. For other data types, you can use the type conversion functions provided by GaussDB(DWS), such as the CAST function, to forcibly convert them.

Table 9-1 lists common implicit data type conversions in GaussDB(DWS).

NOTICE

The valid value range of DATE supported by GaussDB(DWS) is from 4713 B.C. to 294276 A.D.

Table 9-1 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	-

9.3 GaussDB(DWS) Stored Procedure Array and Record

9.3.1 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. Run the following statement:

```
TYPE array_type IS VARRAY(size) OF data_type [NOT NULL];
```

Its parameters are as follows:

- **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 type to be defined. The value is a positive integer.
- **data_type**: indicates the types of members in the array type to be created.
- **NOT NULL**: an optional constraint. It can be used to ensure that none of the elements in the array is **NULL**.

NOTE

- In GaussDB(DWS), an array automatically increases. If an access violation occurs, a null value will be returned, and no error message will be reported. If out-of-bounds write occurs in an array, the message **Subscript outside of limit** is displayed.
- The scope of an array type defined in a stored procedure takes effect only in this storage process.
- 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(DWS) prefers the array type defined in a stored procedure to declare array variables.

In GaussDB(DWS) 8.1.0 and earlier versions, the system does not verify the length of array elements and out-of-bounds write because the array can automatically increase. This version adds related constraints to be compatible with Oracle databases. If out-of-bounds write exists, you can configure **varray_verification** in the parameter **behavior_compat_options** to be compatible with previously unverified operations.

Example:

```
-- Declare an array in a stored procedure.
CREATE OR REPLACE PROCEDURE array_proc
AS
      TYPE ARRAY_INTEGER IS VARRAY(1024) OF INTEGER;--Define the array type.
      TYPE ARRAY_INTEGER_NOT_NULL IS VARRAY(1024) OF INTEGER NOT NULL;-- Defines non-null array
types.
      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;
      DBMS_OUTPUT.PUT_LINE(ARRINT.COUNT);
      DBMS_OUTPUT.PUT_LINE(ARRINT(1));
```

```
    DBMS_OUTPUT.PUT_LINE(ARRINT(10));
    DBMS_OUTPUT.PUT_LINE(ARRINT(ARRINT.FIRST));
    DBMS_OUTPUT.PUT_LINE(ARRINT(ARRINT.last));
END;
/

-- Invoke the stored procedure.
CALL array_proc();
10
1
10
1
10
-- Delete the stored procedure.
DROP PROCEDURE array_proc;
```

Declaration and Use of Rowtype Arrays

In addition to the declaration and use of common arrays and non-null arrays in the preceding example, the array also supports the declaration and use of rowtype arrays.

Example:

```
-- Use the COUNT function on an array in a stored procedure.
CREATE TABLE tbl (a int, b int);
INSERT INTO tbl VALUES(1, 2),(2, 3),(3, 4);
CREATE OR REPLACE PROCEDURE array_proc
AS
  CURSOR all_tbl IS SELECT * FROM tbl ORDER BY a;
  TYPE tbl_array_type IS varray(50) OF tbl%rowtype; -- Defines the array of the rowtype type. tbl indicates 
any table.
  tbl_array tbl_array_type;
  tbl_item tbl%rowtype;
  inx1 int;
BEGIN
  tbl_array := tbl_array_type();
  inx1 := 0;
  FOR tbl_item IN all_tbl LOOP
    inx1 := inx1 + 1;
    tbl_array(inx1) := tbl_item;
  END LOOP;
  WHILE inx1 IS NOT NULL LOOP
    DBMS_OUTPUT.PUT_LINE('tbl_array(inx1).a=' || tbl_array(inx1).a || ' tbl_array(inx1).b=' || 
tbl_array(inx1).b);
    inx1 := tbl_array.PRIOR(inx1);
  END LOOP;
END;
/
```

The execution output is as follows:

```
call array_proc();
tbl_array(inx1).a=3 tbl_array(inx1).b=4
tbl_array(inx1).a=2 tbl_array(inx1).b=3
tbl_array(inx1).a=1 tbl_array(inx1).b=2
```

Array Related Functions

GaussDB(DWS) supports Oracle-related array functions. You can use the following functions to obtain array attributes or perform operations on the array content.

COUNT

Returns the number of elements in the current array. Only the initialized elements or the elements extended by the EXTEND function are counted.

Use:

varray.**COUNT** or *varray*.**COUNT()**

Example:

```
-- Use the COUNT function on an array in a stored procedure.
CREATE OR REPLACE PROCEDURE test_varray
AS
    TYPE varray_type IS VARRAY(20) OF INT;
    v_varray varray_type;
BEGIN
    v_varray := varray_type(1, 2, 3);
    DBMS_OUTPUT.PUT_LINE('v_varray.count=' || v_varray.count);
    v_varray.extend;
    DBMS_OUTPUT.PUT_LINE('v_varray.count=' || v_varray.count);
END;
/
```

The execution output is as follows:

```
call test_varray();
v_varray.count=3
v_varray.count=4
```

FIRST and LAST

The FIRST function can return the subscript of the first element. The LAST function can return the subscript of the last element.

Use:

varray.**FIRST** or *varray*.**FIRST()**

varray.**LAST** or *varray*.**LAST()**

Example:

```
-- Use the FIRST and LAST functions on an array in a stored procedure.
CREATE OR REPLACE PROCEDURE test_varray
AS
    TYPE varray_type IS VARRAY(20) OF INT;
    v_varray varray_type;
BEGIN
    v_varray := varray_type(1, 2, 3);
    DBMS_OUTPUT.PUT_LINE('v_varray.first=' || v_varray.first);
    DBMS_OUTPUT.PUT_LINE('v_varray.last=' || v_varray.last);
END;
/
```

The execution output is as follows:

```
call test_varray();
v_varray.first=1
v_varray.last=3
```

EXTEND

NOTE

The EXTEND function is used to be compatible with two Oracle database operations. In GaussDB(DWS), an array automatically grows, and the EXTEND function is not necessary. For a newly written stored procedure, you do not need to use the EXTEND function.

The EXTEND function can extend arrays. The EXTEND function can be invoked in either of the following ways:

- Method 1:

  EXTEND contains an integer input parameter, indicating that the array size is extended by the specified length. After executing the EXTEND function, the values of the COUNT and LAST functions change accordingly.

  Use:

  varray.EXTEND(size)

  By default, one bit is added to the end of *varray***.EXTEND**, which is equivalent to *varray***.EXTEND(1)**.

- Method 2:

  EXTEND contains two integer input parameters. The first parameter indicates the length of the extended size. The second parameter indicates that the value of the extended array element is the same as that of the element with the **index** subscript.

  Use:

  varray.EXTEND(size, index)

Example:

```
-- Use the EXTEND function on an array in a stored procedure.
CREATE OR REPLACE PROCEDURE test_varray
AS
    TYPE varray_type IS VARRAY(20) OF INT;
    v_varray varray_type;
BEGIN
    v_varray := varray_type(1, 2, 3);
    v_varray.extend(3);
    DBMS_OUTPUT.PUT_LINE('v_varray.count=' || v_varray.count);
    v_varray.extend(2,3);
    DBMS_OUTPUT.PUT_LINE('v_varray.count=' || v_varray.count);
    DBMS_OUTPUT.PUT_LINE('v_varray(7)=' || v_varray(7));
    DBMS_OUTPUT.PUT_LINE('v_varray(8)=' || v_varray(7));
END;
/
```

The execution output is as follows:

```
call test_varray();
v_varray.count=6
v_varray.count=8
v_varray(7)=3
v_varray(8)=3
```

NEXT and PRIOR

The NEXT and PRIOR functions are used for cyclic array traversal. The NEXT function returns the subscript of the next array element based on the input parameter **index**. If the subscript reaches the maximum value, **NULL** is returned. The PRIOR function returns the subscript of the previous array element based on the input parameter **index**. If the minimum value of the array subscript is reached, **NULL** is returned.

Use:

varray.NEXT(index)

varray.PRIOR(index)

Example:

```
-- Use the NEXT and PRIOR functions on an array in a stored procedure.
CREATE OR REPLACE PROCEDURE test_varray
AS
    TYPE varray_type IS VARRAY(20) OF INT;
    v_varray varray_type;
    i int;
BEGIN
    v_varray := varray_type(1, 2, 3);

    i := v_varray.COUNT;
    WHILE i IS NOT NULL LOOP
        DBMS_OUTPUT.PUT_LINE('test prior v_varray('||i||')=' || v_varray(i));
        i := v_varray.PRIOR(i);
    END LOOP;

    i := 1;
    WHILE i IS NOT NULL LOOP
        DBMS_OUTPUT.PUT_LINE('test next v_varray('||i||')=' || v_varray(i));
        i := v_varray.NEXT(i);
    END LOOP;
END;
/
```

The execution output is as follows:

```
call test_varray();
test prior v_varray(3)=3
test prior v_varray(2)=2
test prior v_varray(1)=1
test next v_varray(1)=1
test next v_varray(2)=2
test next v_varray(3)=3
```

EXISTS

Determines whether an array subscript exists.

Use:

varray.EXISTS(index)

Example:

```
-- Use the EXISTS function on an array in a stored procedure.
CREATE OR REPLACE PROCEDURE test_varray
AS
    TYPE varray_type IS VARRAY(20) OF INT;
    v_varray varray_type;
BEGIN
    v_varray := varray_type(1, 2, 3);
    IF v_varray.EXISTS(1) THEN
        DBMS_OUTPUT.PUT_LINE('v_varray.EXISTS(1)');
    END IF;
    IF NOT v_varray.EXISTS(10) THEN
        DBMS_OUTPUT.PUT_LINE('NOT v_varray.EXISTS(10)');
    END IF;
END;
/
```

The execution output is as follows:

```
call test_varray();
v_varray.EXISTS(1)
NOT v_varray.EXISTS(10)
```

TRIM

Deletes a specified number of elements from the end of an array.

Use:

varray.TRIM(size)

varray.**TRIM** is equivalent to *varray*.**TRIM(1)**, because the default input parameter is **1**.

Example:

```
-- Use the TRIM function on an array in a stored procedure.
CREATE OR REPLACE PROCEDURE test_varray
AS
    TYPE varray_type IS VARRAY(20) OF INT;
    v_varray varray_type;
BEGIN
    v_varray := varray_type(1, 2, 3, 4, 5);
    v_varray.trim(3);
    DBMS_OUTPUT.PUT_LINE('v_varray.count' || v_varray.count);
    v_varray.trim;
    DBMS_OUTPUT.PUT_LINE('v_varray.count:' || v_varray.count);
END;
/
```

The execution output is as follows:

```
call test_varray();
v_varray.count:2
v_varray.count:1
```

DELETE

Deletes all elements from an array.

Use:

varray.**DELETE** or *varray*.**DELETE()**

Example:

```
-- Use the DELETE function on an array in a stored procedure.
CREATE OR REPLACE PROCEDURE test_varray
AS
    TYPE varray_type IS VARRAY(20) OF INT;
    v_varray varray_type;
BEGIN
    v_varray := varray_type(1, 2, 3, 4, 5);
    v_varray.delete;
    DBMS_OUTPUT.PUT_LINE('v_varray.count:' || v_varray.count);
END;
/
```

The execution output is as follows:

```
call test_varray();
v_varray.count:0
```

LIMIT

Returns the allowed maximum length of an array.

Use:

varray.**LIMIT** or *varray*.**LIMIT()**

Example:

```
-- Use the LIMIT function on an array in a stored procedure.
CREATE OR REPLACE PROCEDURE test_varray
AS
    TYPE varray_type IS VARRAY(20) OF INT;
    v_varray varray_type;
BEGIN
    v_varray := varray_type(1, 2, 3, 4, 5);
    DBMS_OUTPUT.PUT_LINE('v_varray.limit:' || v_varray.limit);
END;
/
```

The execution output is as follows:

```
call test_varray();
v_varray.limit:20
```

9.3.2 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 9-1**.

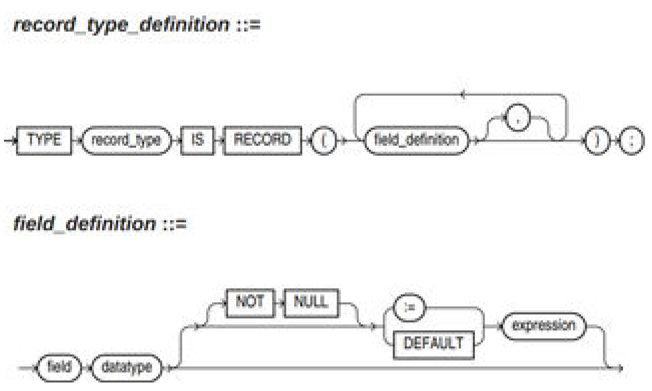


Figure 9-1 Syntax of the record type

The syntax is described as follows:

- **record_type**: record name
- **field**: record columns
- **datatype**: record data type
- **expression**: expression for setting a default value

NOTE

- 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.

Examples

```
The table used in the following stored procedure is defined as follows:
CREATE TABLE emp_rec
(
    empno            numeric(4,0),
    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)
)
with (orientation = column,compression=middle)
distribute by hash (sal);
\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 stored procedure.
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;

-- Invoke the stored procedure.
CALL regress_record('abc');
INFO:  employer name: WARD , epno:18
INFO:  employer1 name: WARD , epno: 18
INFO:  employer1 name: <NULL> , epno: <NULL>
INFO:  employer2 name: SCOTT ,epno: 10
-- Delete the stored procedure.
DROP PROCEDURE regress_record;
```

9.4 GaussDB(DWS) Stored Procedure Declaration Syntax

Basic 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. It is mandatory.
  ```
  BEGIN
  ```
- **EXCEPTION**: processes errors. It is 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 parameter **-r** is executed using the **gsql** tool.

PL/SQL Block Classification

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 **Anonymous Block**.
- Subprogram: a stored procedure, function, operator, or packages stored in a database. A subprogram created in a database can be called by other programs.

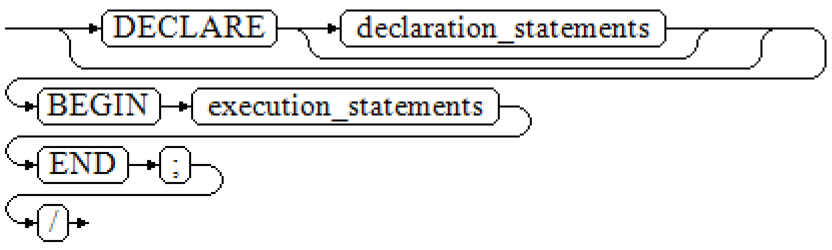
Anonymous Block

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 9-2 shows the syntax diagrams for an anonymous block.

Figure 9-2 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.

Examples

The following lists basic anonymous block programs:

```
-- Null statement block:
BEGIN
     NULL;
END;
/

-- Print information to the console:
BEGIN
     dbms_output.put_line('hello world!');
END;
/

-- Print variable contents to the console:
DECLARE
     my_var VARCHAR2(30);
BEGIN
     my_var :='world';
     dbms_output.put_line('hello'||my_var);
END;
/
```

Subprogram

A subprogram stores stored procedures, functions, operators, and advanced packages. A subprogram created in a database can be called by other programs.

9.5 Basic Statements of GaussDB(DWS) Stored Procedures

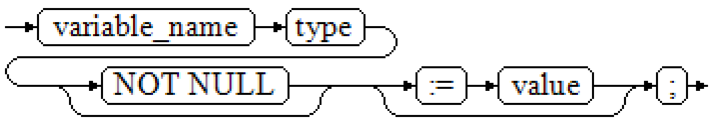
Variable Definition Statement

This section describes the declaration of variables in the PL/SQL and the scope of this variable in codes.

Variable declaration

For the variable declaration syntax, see **Figure 9-3**.

Figure 9-3 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

```
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 that has the same data type as 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** even if you do not know 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.

%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
```

NOTICE

If multiple CNs are used, the **%ROWTYPE** and **%TYPE** attributes of temporary tables cannot be declared in a stored procedure, because a temporary table is valid only in the current session and is invisible to other CNs in the compilation phase. In this case, a message is displayed indicating that the temporary table does not exist.

Variable scope

The scope of a variable indicates the accessibility and availability of a 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.

Examples

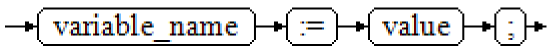
```
DECLARE
    emp_id  INTEGER :=7788; -- Define a variable and assign a value to it.
    outer_var  INTEGER :=6688; -- Define a variable and assign a value to it.
BEGIN
    DECLARE
        emp_id INTEGER :=7799; -- Define a variable and assign a value to it.
        inner_var  INTEGER :=6688; -- Define a variable and assign a value to it.
    BEGIN
        dbms_output.put_line('inner emp_id ='||emp_id); -- Display the value as 7799.
        dbms_output.put_line('outer_var ='||outer_var); -- Cite variables of an outer block.
    END;
    dbms_output.put_line('outer emp_id ='||emp_id); -- Display the value as 7788.
END;
/
```

Assignment Statement

Syntax

Figure 9-4 shows the syntax diagram for assigning a value to a variable.

Figure 9-4 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**.

Examples

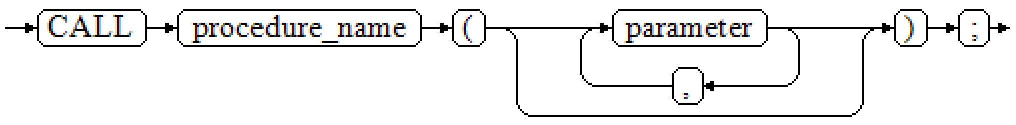
```
DECLARE
    emp_id  INTEGER := 7788;-- Value assignment
BEGIN
    emp_id := 5;-- Value assignment
    emp_id := 5*7784;
END;
/
```

Call Statement

Syntax

Figure 9-5 shows the syntax diagram for calling a clause.

Figure 9-5 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:
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 staffs where section_id = section;
END;
/

-- Create the stored procedure proc_return:
CREATE OR REPLACE PROCEDURE proc_return
AS
v_num NUMBER(8,2);
v_sum INTEGER;
BEGIN
proc_staffs(30, v_sum, v_num);  --Call a statement.
dbms_output.put_line(v_sum||'#'||v_num);
RETURN;   --Return a statement.
END;
/

-- Invoke a stored procedure proc_return:
CALL proc_return();

-- Delete a stored procedure:
DROP PROCEDURE proc_staffs;
DROP PROCEDURE proc_return;

--Create the function func_return.
CREATE OR REPLACE FUNCTION func_return returns void
language plpgsql
AS $$
DECLARE
v_num INTEGER := 1;
BEGIN
dbms_output.put_line(v_num);
RETURN;   --Return a statement.
END $$;


-- Invoke the function func_return.
CALL func_return();
1

-- Delete the function:
DROP FUNCTION func_return;
```

9.6 Dynamic Statements of GaussDB(DWS) Stored Procedures

9.6.1 Executing Dynamic Query Statements

You can perform dynamic queries using **EXECUTE IMMEDIATE** or **OPEN FOR** in GaussDB(DWS). **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 9-6 shows the syntax diagram.

Figure 9-6 EXECUTE IMMEDIATE dynamic_select_clause::=

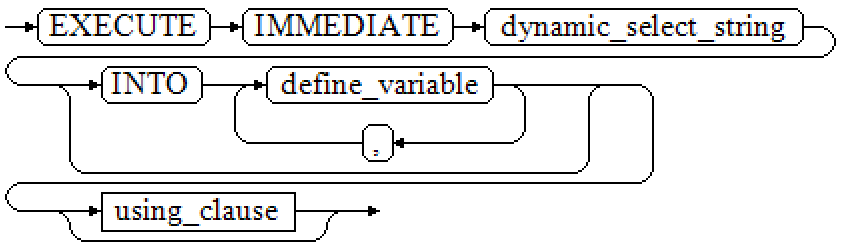
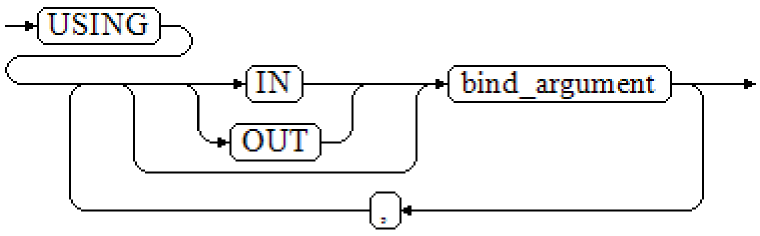


Figure 9-7 shows the syntax diagram for **using_clause**.

Figure 9-7 using_clause-1



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.

Example

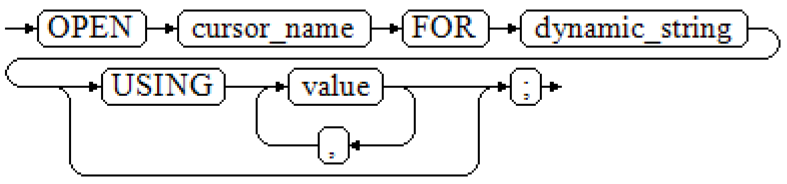
```
--Retrieve values from dynamic statements (INTO clause).
DECLARE
   staff_count  VARCHAR2(20);
BEGIN
   EXECUTE IMMEDIATE 'select count(*) from staffs'
      INTO staff_count;
   dbms_output.put_line(staff_count);
END;
/

--Pass and retrieve values (the INTO clause is used before the USING clause).
CREATE OR REPLACE PROCEDURE dynamic_proc
AS
   staff_id     NUMBER(6) := 200;
   first_name   VARCHAR2(20);
   salary       NUMBER(8,2);
BEGIN
   EXECUTE IMMEDIATE 'select first_name, salary from staffs where staff_id = :1'
       INTO first_name, salary
       USING IN staff_id;
   dbms_output.put_line(first_name || ' ' || salary);
END;
/

-- Invoke the stored procedure.
CALL dynamic_proc();

-- Delete the stored procedure.
DROP PROCEDURE dynamic_proc;
```

OPEN FOR

Dynamic query statements can be executed by using **OPEN FOR** to open dynamic cursors.

For details about the syntax, see **Figure 9-8**.

Figure 9-8 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 **GaussDB(DWS) Stored Procedure Cursor**.

Example

```
DECLARE
    name          VARCHAR2(20);
    phone_number  VARCHAR2(20);
    salary        NUMBER(8,2);
    sqlstr        VARCHAR2(1024);

    TYPE app_ref_cur_type IS REF CURSOR; -- Define the cursor type.
    my_cur app_ref_cur_type; -- Define the cursor variable.

BEGIN
    sqlstr := 'select first_name,phone_number,salary from staffs
         where section_id = :1';
    OPEN my_cur FOR sqlstr USING '30'; -- Open the cursor. using is optional.
    FETCH my_cur INTO name, phone_number, salary; -- Retrieve the data.
    WHILE my_cur%FOUND LOOP
          dbms_output.put_line(name||'#'||phone_number||'#'||salary);
          FETCH my_cur INTO name, phone_number, salary;
    END LOOP;
    CLOSE my_cur; -- Close the cursor.
END;
/
```

9.6.2 Executing Dynamic Non-query Statements

Syntax

Figure 9-9 shows the syntax diagram.

Figure 9-9 noselect::=

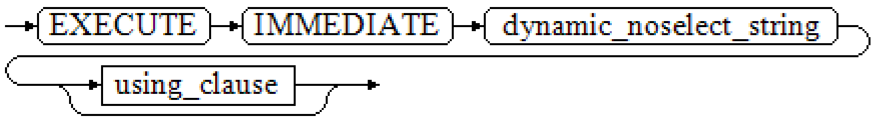
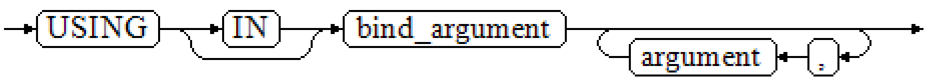


Figure 9-10 shows the syntax diagram for **using_clause**.

Figure 9-10 using_clause-2



The above syntax diagram is explained as follows:

USING IN bind_argument is used to specify the variable that transfers values to dynamic SQL statements. It 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:
CREATE TABLE sections_t1
(
   section       NUMBER(4) ,
   section_name  VARCHAR2(30),
   manager_id    NUMBER(6),
   place_id      NUMBER(4)
)
DISTRIBUTE BY hash(manager_id);

--Declare a variable:
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:
SELECT * FROM sections_t1;

--Delete the table.
DROP TABLE sections_t1;
```

9.6.3 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 9-11 shows the syntax diagram.

Figure 9-11 call_procedure::=

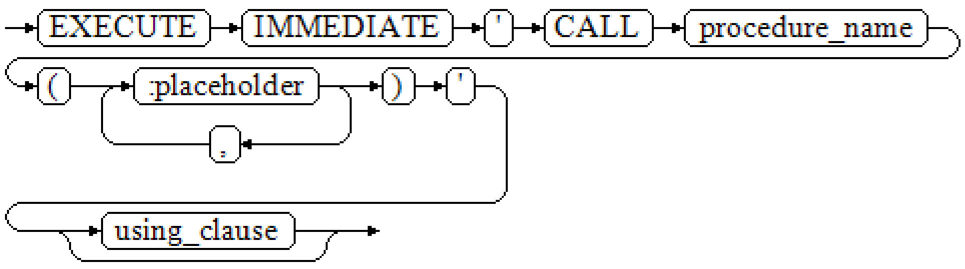
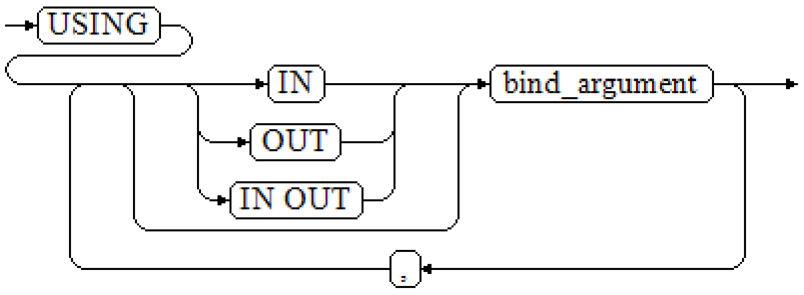


Figure 9-12 shows the syntax diagram for **using_clause**.

Figure 9-12 using_clause-3



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 the 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.

Examples

```
--Create the stored procedure proc_add:
CREATE OR REPLACE PROCEDURE proc_add
(
    param1    in   INTEGER,
    param2    out  INTEGER,
    param3    in   INTEGER
)
AS
```

```
BEGIN
   param2:= param1 + param3;
END;
/

DECLARE
    input1 INTEGER:=1;
    input2 INTEGER:=2;
    statement  VARCHAR2(200);
    param2     INTEGER;
BEGIN
   --Declare the call statement:
    statement := 'call proc_add(:col_1, :col_2, :col_3)';
   --Execute the statement:
    EXECUTE IMMEDIATE statement
        USING IN input1, OUT param2, IN input2;
    dbms_output.put_line('result is: '||to_char(param2));
END;
/

-- Delete the stored procedure.
DROP PROCEDURE proc_add;
```

9.6.4 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 9-13 shows the syntax diagram.

Figure 9-13 call_anonymous_block::=

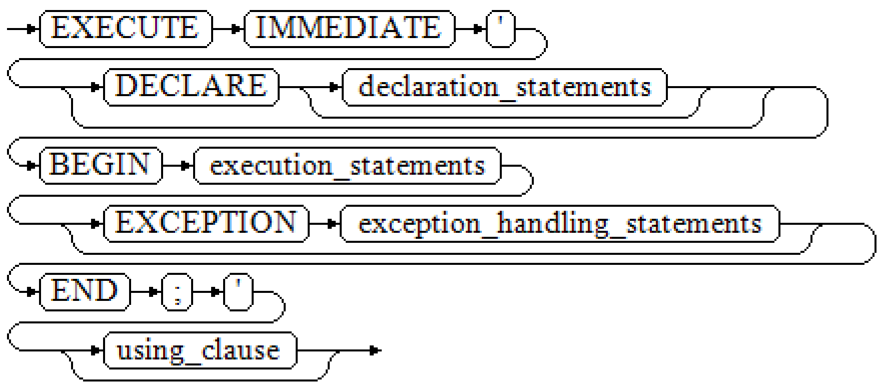
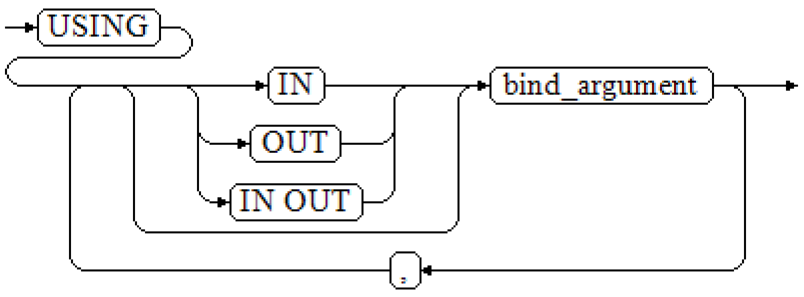


Figure 9-14 shows the syntax diagram for **using_clause**.

Figure 9-14 using_clause-4



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 the parameters are the same. The sequences of the parameters corresponding to the placeholders and the USING parameters are the same.
- Currently in GaussDB(DWS), when dynamic statements call anonymous blocks, placeholders cannot be used to pass input and output parameters in an **EXCEPTION** statement.

Example

```
--Create the stored procedure dynamic_proc.
CREATE OR REPLACE PROCEDURE dynamic_proc
AS
   staff_id     NUMBER(6) := 200;
   first_name   VARCHAR2(20);
   salary       NUMBER(8,2);
BEGIN
--Execute the anonymous block.
    EXECUTE IMMEDIATE 'begin select first_name, salary into :first_name, :salary from staffs where
staff_id= :dno; end;'
       USING OUT first_name, OUT salary, IN staff_id;
   dbms_output.put_line(first_name|| ' ' || salary);
END;
/

-- Invoke the stored procedure.
CALL dynamic_proc();

-- Delete the stored procedure.
DROP PROCEDURE dynamic_proc;
```

9.7 GaussDB(DWS) Stored Procedure Control Statements

9.7.1 RETURN Statements

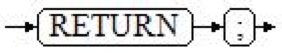
GaussDB(DWS) provides two methods for returning data: **RETURN** (or **RETURN NEXT**) and **RETURN QUERY**. **RETURN NEXT** and **RETURN QUERY** are used only for functions and cannot be used for stored procedures.

RETURN

Syntax

Figure 9-15 shows the syntax diagram for a return statement.

Figure 9-15 return_clause::=



The syntax details are as follows:

This statement returns control from a stored procedure or function to a caller.

Examples

```
-- Create the stored procedure proc_staffs:
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 staffs where section_id = section;
END;
/

-- Create the stored procedure proc_return:
CREATE OR REPLACE PROCEDURE proc_return
AS
v_num NUMBER(8,2);
v_sum INTEGER;
BEGIN
proc_staffs(30, v_sum, v_num);  --Call a statement.
dbms_output.put_line(v_sum||'#'||v_num);
RETURN;   --Return a statement.
END;
/

-- Invoke a stored procedure proc_return:
CALL proc_return();

-- Delete a stored procedure:
DROP PROCEDURE proc_staffs;
DROP PROCEDURE proc_return;

--Create the function func_return.
CREATE OR REPLACE FUNCTION func_return returns void
language plpgsql
AS $$
DECLARE
v_num INTEGER := 1;
BEGIN
dbms_output.put_line(v_num);
RETURN;   --Return a statement.
```

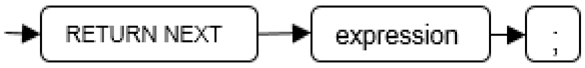
```
END $$;


-- Invoke the function func_return.
CALL func_return();
1

-- Delete the function:
DROP FUNCTION func_return;
```

RETURN NEXT and RETURN QUERY

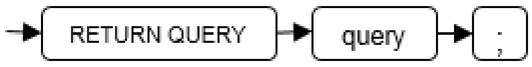
Syntax

When creating a function, specify **SETOF datatype** for the return values.

return_next_clause::=



return_query_clause::=



The syntax details are 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 **USING**.

Examples

```
CREATE TABLE t1(a int);
INSERT INTO t1 VALUES(1),(10);

--RETURN NEXT
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;
call fun_for_return_next();
 a
---
 1
 10
(2 rows)

-- RETURN QUERY
```

```
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;
call fun_for_return_next();
 a
---
 1
 10
(2 rows)
```

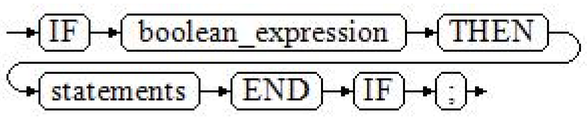
9.7.2 Conditional Statements

Conditional statements are used to decide whether given conditions are met. Operations are executed based on the decisions made.

GaussDB(DWS) supports five usages of **IF**:

- IF_THEN

  Figure 9-16 IF_THEN::=

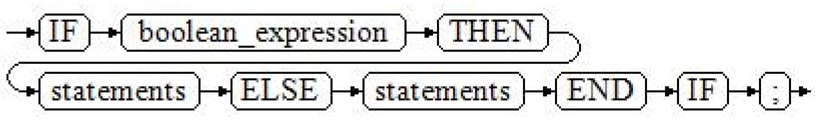
  IF_THEN is the simplest form of **IF**. If the condition is true, statements are executed. If it is false, they are skipped.

  Example

  ```
  IF v_user_id <> 0 THEN
      UPDATE users SET email = v_email WHERE user_id = v_user_id;
  END IF;
  ```

- IF_THEN_ELSE

  Figure 9-17 IF_THEN_ELSE::=

  IF-THEN-ELSE statements add **ELSE** branches and can be executed if the condition is **false**.

  Example

  ```
  IF parentid IS NULL OR parentid = ''
  THEN
      RETURN;
  ELSE
      hp_true_filename(parentid); -- Call the stored procedure.
  END IF;
  ```

- IF_THEN_ELSE IF

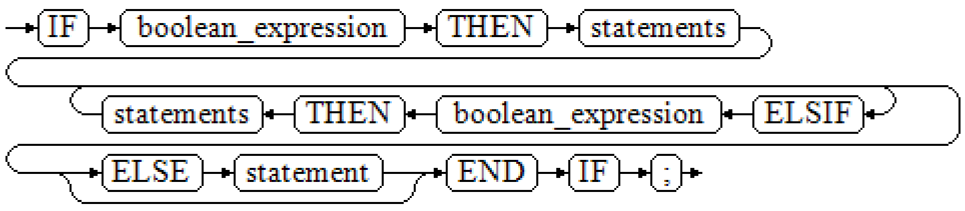
  IF statements can be nested in the following way:

```
IF gender = 'm' THEN
    pretty_gender := 'man';
ELSE
    IF gender = 'f' THEN
        pretty_gender := '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 9-18 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

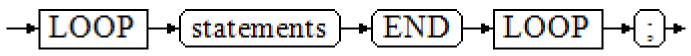
```
CREATE OR REPLACE PROCEDURE proc_control_structure(i in integer)
AS
    BEGIN
        IF i > 0 THEN
            raise info 'i:% is greater than 0. ',i;
        ELSIF i < 0 THEN
            raise info 'i:% is smaller than 0. ',i;
        ELSE
            raise info 'i:% is equal to 0. ',i;
        END IF;
        RETURN;
    END;
/

CALL proc_control_structure(3);

-- Delete the stored procedure:
DROP PROCEDURE proc_control_structure;
```

9.7.3 Loop Statements

Simple LOOP Statements

The syntax diagram is as follows.

Figure 9-19 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);
```

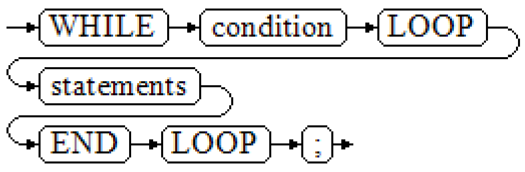
NOTICE

The loop must be exploited together with **EXIT**; otherwise, a dead loop occurs.

WHILE-LOOP Statements

The syntax diagram is as follows.

Figure 20 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.

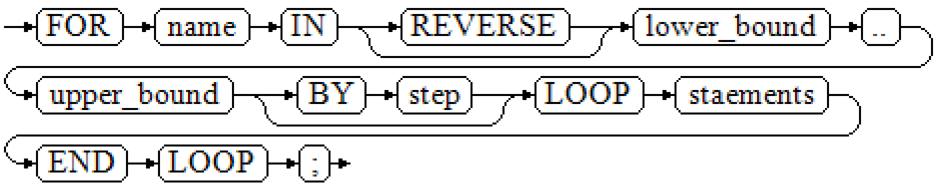
Examples

```
CREATE TABLE integertable(c1 integer) DISTRIBUTE BY hash(c1);
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*

The syntax diagram is as follows.

Figure 9-21 for_loop::=



NOTE

- The variable **name** is automatically defined as the **integer** type and exists only in this loop. The variable name falls between lower_bound and upper_bound.
- When the keyword **REVERSE** is used, the lower bound must be greater than or equal to the upper bound; otherwise, the loop body is not executed.

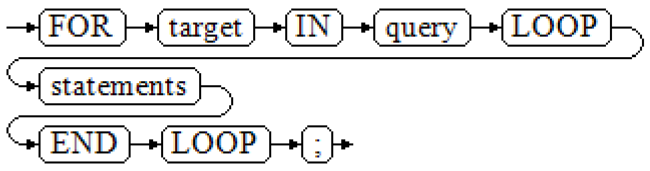
Example:

```
-- Loop from 0 to 5:
CREATE OR REPLACE PROCEDURE proc_for_loop()
AS
    BEGIN
    FOR I IN 0..5 LOOP
        DBMS_OUTPUT.PUT_LINE('It is '||to_char(I) || ' time;') ;
    END LOOP;
END;
/

-- Invoke a function:
CALL proc_for_loop();

-- Delete the stored procedure:
DROP PROCEDURE proc_for_loop;
```

FOR_LOOP Query Statements

The syntax diagram is as follows.

Figure 9-22 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 target value is the query result.

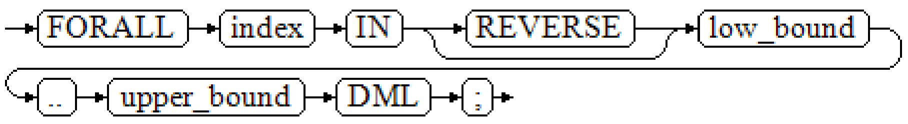
Example:

```
-- Display the query result from the loop:
CREATE OR REPLACE PROCEDURE proc_for_loop_query()
AS
    record VARCHAR2(50);
BEGIN
    FOR record IN SELECT spcname FROM pg_tablespace LOOP
    dbms_output.put_line(record);
    END LOOP;
END;
/

-- Invoke a function.
CALL proc_for_loop_query();

-- Delete the stored procedure.
DROP PROCEDURE proc_for_loop_query;
```

FORALL Batch Query Statements

The syntax diagram is as follows.

Figure 9-23 forall::=



NOTE

The variable **index** is automatically defined as the **integer** type and exists only in this loop. The index value falls between low_bound and upper_bound.

Example:

```
CREATE TABLE hdfs_t1 (
  title NUMBER(6),
 did VARCHAR2(20),
 data_peroid VARCHAR2(25),
 kind VARCHAR2(25),
 interval VARCHAR2(20),
 time DATE,
 isModified VARCHAR2(10)
)
DISTRIBUTE BY hash(did);

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
        insert into hdfs_t1(title) values(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;
```

9.7.4 Branch Statements

Syntax

Figure 9-24 shows the syntax diagram.

Figure 9-24 case_when::=

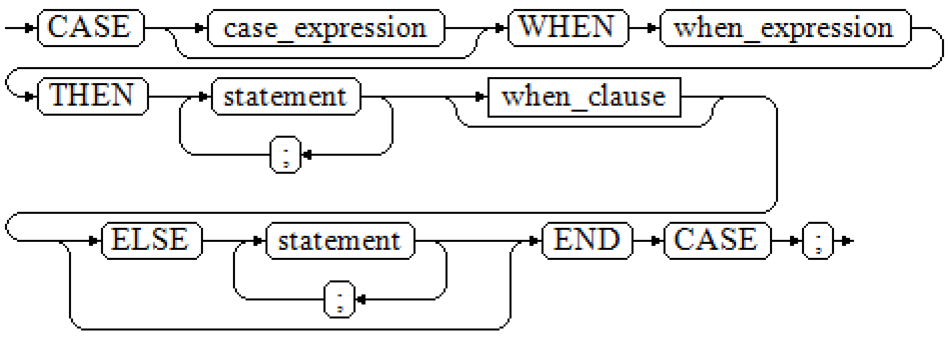
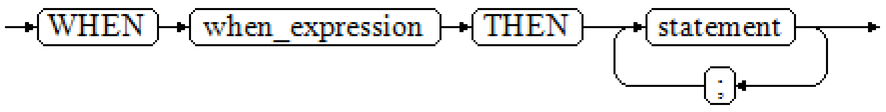


Figure 9-25 shows the syntax diagram for **when_clause**.

Figure 9-25 when_clause::=



Parameter description:

- **case_expression**: specifies the variable or expression.
- **when_expression**: specifies the constant or conditional expression.
- **statement**: specifies the statement to execute.

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;
```

9.7.5 NULL Statements

In PL/SQL programs, a NULL statement can be used to indicate "do nothing", which is also known as an empty statement.

A NULL statement acts as a placeholder and can give meaning to certain statements, improving the readability of the program.

Syntax

Here are some examples of how to use NULL statements.

```
DECLARE
    ...
BEGIN
    ...
    IF v_num IS NULL THEN
        NULL; --No data needs to be processed.
    END IF;
END;
/
```

9.7.6 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 inside the executed statement, the statement rolls back and goes to the EXCEPTION list to find the first condition that matches the error. 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:

The error can be caught by an enclosing block with **EXCEPTION**, or if there is none it aborts processing of the function.

The *condition* name 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)) DISTRIBUTE BY hash(id);

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:

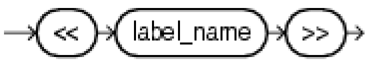
```
CREATE TABLE db (a INT, b TEXT);

CREATE FUNCTION merge_db(key INT, data TEXT) RETURNS VOID AS
$$
BEGIN
    LOOP

-- Try updating 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, there could be a
unique-key failure.
        BEGIN
            INSERT INTO db(a,b) VALUES (key, data);
            RETURN;
        EXCEPTION WHEN unique_violation THEN
        -- 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 ;
```

9.7.7 GOTO Statements

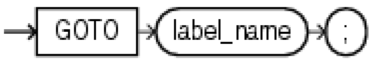
The **GOTO** statement unconditionally transfers the control from the current statement to a labeled statement. The **GOTO** statement changes the execution logic. Therefore, use this statement only when necessary. Alternatively, you can use the **EXCEPTION** statement to handle issues in special scenarios. To run the **GOTO** statement, the labeled statement must be unique.

Syntax

label declaration ::=



goto statement ::=



Examples

```
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();
DROP PROCEDURE GOTO_test();
```

Constraints

The **GOTO** statement has the following constraints:

- The **GOTO** statement does not allow multiple labeled statements even if they are in different blocks.
  ```
  BEGIN
    GOTO pos1;
    <<pos1>>
    SELECT * FROM ...
    <<pos1>>
    UPDATE t1 SET ...
  END;
  ```
- The **GOTO** statement cannot transfer control to the **IF**, **CASE**, or **LOOP** statement.
  ```
  BEGIN
     GOTO pos1;
     IF valid THEN
       <<pos1>>
       SELECT * FROM ...
     END IF;
   END;
  ```

- The **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;
```

- The **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;
```

- The **GOTO** statement cannot transfer control from an **EXCEPTION** block to the current **BEGIN-END** block but can transfer to an outer **BEGIN-END** block.

```
BEGIN
   <<pos1>>
   UPDATE t1 SET ...
   EXCEPTION
     WHEN condition THEN
        GOTO pos1;
 END;
```

- If the labeled statement in the **GOTO** statement does not exist, you need to 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;
/
```

9.8 Other Statements in a GaussDB(DWS) Stored Procedure

Lock Operations

GaussDB(DWS) provides multiple lock modes to control concurrent accesses to table data. These modes are used when Multi-Version Concurrency Control (MVCC) cannot give expected behaviors. Alike, most GaussDB(DWS) 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.

Cursor Operations

GaussDB(DWS) 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 them 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**.

9.9 GaussDB(DWS) Stored Procedure Cursor

9.9.1 Overview

To process SQL statements, the stored procedure process assigns a memory segment to store context association. Cursors are handles or pointers to context areas. With cursors, stored procedures can control alterations in context areas.

NOTICE

If JDBC is used to call a stored procedure whose returned value is a cursor, the returned cursor is not available.

Cursors are classified into explicit cursors and implicit cursors. **Table 9-2** shows the usage conditions of explicit and implicit cursors for different SQL statements.

Table 9-2 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

9.9.2 Explicit Cursor

An explicit cursor is used to process query statements, particularly when the query results contain multiple records.

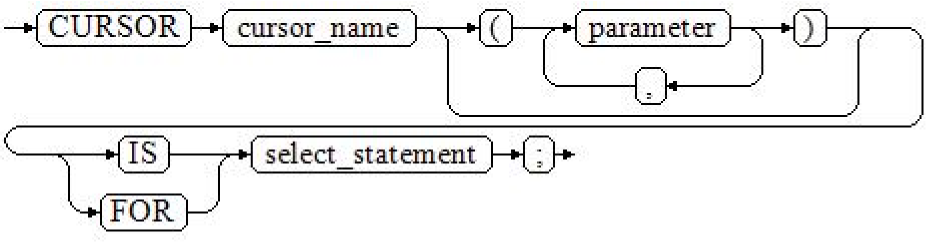
Procedure

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 9-26 shows the syntax diagram for defining a static cursor.

Figure 9-26 static_cursor_define::=



Parameter description:

- **cursor_name**: defines a cursor name.
- **parameter**: specifies cursor parameters. Only input parameters are allowed in the following format:

```
parameter_name datatype
```

- **select_statement**: specifies a query statement.

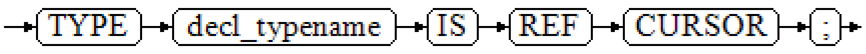
NOTE

The system automatically determines whether the cursor can be used for backward fetches 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.

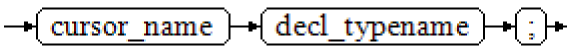
Figure 9-27 and **Figure 9-28** show the syntax diagrams for defining a dynamic cursor.

Figure 9-27 cursor_typename::=



GaussDB(DWS) supports the dynamic cursor type **sys_refcursor**. A function or stored procedure can use the **sys_refcursor** parameter to pass on or pass out the cursor result set. A function can return **sys_refcursor** to return the cursor result set.

Figure 9-28 dynamic_cursor_define::=

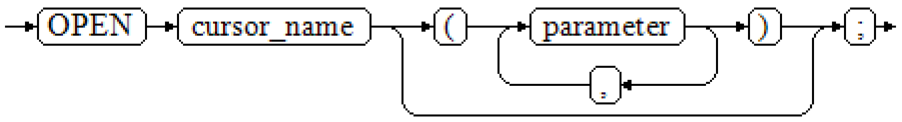


Step 2 **Open the static cursor:** Execute the **SELECT** statement corresponding to the cursor. The query result is placed in the work area and the pointer directs to the

head of the work area to identify the cursor result set. If the cursor query statement contains the **FOR UPDATE** option, the **OPEN** statement locks the data row corresponding to the cursor result set in the database table.

Figure 9-29 shows the syntax diagram for opening a static cursor.

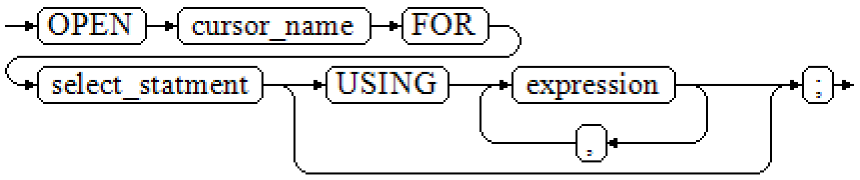
Figure 9-29 open_static_cursor::=



Open the dynamic cursor: Use the **OPEN FOR** statement to open the dynamic cursor and the SQL statement is dynamically bound.

Figure 9-30 shows the syntax diagram for opening a dynamic cursor.

Figure 9-30 open_dynamic_cursor::=

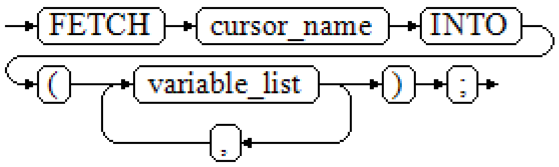


A PL/SQL program cannot use the **OPEN** statement to repeatedly open a cursor.

Step 3 **Fetch cursor data**: Retrieve data rows in the result set and place them in specified output variables.

Figure 9-31 shows the syntax diagram for fetching cursor data.

Figure 9-31 fetch_cursor::=



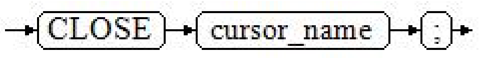
Step 4 Process the record.

Step 5 Continue to process until the active set has no record.

Step 6 **Close the cursor**: When fetching and finishing the data in the cursor result set, close the cursor immediately to release system resources used by the cursor and invalidate the work area of the cursor so that the **FETCH** statement cannot be used to fetch data any more. A closed cursor can be reopened using the **OPEN** statement.

Figure 9-32 shows the syntax diagram for closing a cursor.

Figure 9-32 close_cursor::=



----End

Attributes

Cursor attributes are used to control program procedures or learn about 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 sections WHERE section_id <= 50;
    CURSOR C2(sect_id INTEGER) IS
        SELECT section_name, place_id FROM 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;
        DBMS_OUTPUT.PUT_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;
        DBMS_OUTPUT.PUT_LINE(DEPT_NAME||'---'||DEPT_LOC);
    END LOOP;
    CLOSE C2;

    SQL_STR := 'SELECT section_name, place_id FROM 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;
        DBMS_OUTPUT.PUT_LINE(DEPT_NAME||'---'||DEPT_LOC);
    END LOOP;
    CLOSE C3;
END;
/

CALL cursor_proc1();

DROP PROCEDURE cursor_proc1;
-- Increase the salary of employees whose salary is lower than CNY3000 by CNY500:
CREATE TABLE staffs_t1 AS TABLE 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 staffs_t1;
BEGIN
   OPEN C;
   LOOP
      FETCH C INTO V_EMPNO, V_SAL;
      EXIT WHEN C%NOTFOUND;
      IF V_SAL<=3000 THEN
            UPDATE staffs_t1 SET salary =salary + 500 WHERE staff_id = V_EMPNO;
      END IF;
   END LOOP;
   CLOSE C;
END;
/

CALL cursor_proc2();

-- Drop the stored procedure:
DROP PROCEDURE cursor_proc2;
DROP TABLE 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 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;
  DBMS_OUTPUT.PUT_LINE(C1%ROWCOUNT);
  EXIT WHEN C1%NOTFOUND;
END LOOP;
END;
/

-- Drop the stored procedure:
DROP PROCEDURE proc_sys_ref;
```

9.9.3 Implicit Cursor

The system automatically sets implicit cursors for non-query statements, such as **ALTER** and **DROP**, and creates work areas for these statements. These implicit cursors are named SQL, which is defined by the system.

Overview

Implicit cursor operations, such as definition, opening, value-grant, and closing, are automatically performed by the system. Users can use only the attributes of implicit cursors to complete operations. The data stored in the work area of an implicit cursor is the latest SQL statement, and is not related to the user-defined explicit cursors.

Format call: SQL%

NOTE

INSERT, **UPDATE**, **DROP**, and **SELECT** statements do not require defined cursors.

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 executed.

Examples

```
-- Delete all employees in a department from the EMP table. If the department has no employees, delete the department from the DEPT table.
CREATE TABLE staffs_t1 AS TABLE staffs;
CREATE TABLE sections_t1 AS TABLE sections;

CREATE OR REPLACE PROCEDURE proc_cursor3()
AS
    DECLARE
    V_DEPTNO NUMBER(4) := 100;
    BEGIN
        DELETE FROM staffs WHERE section_ID = V_DEPTNO;
        -- Proceed based on cursor status:
        IF SQL%NOTFOUND THEN
        DELETE FROM sections_t1 WHERE section_ID = V_DEPTNO;
        END IF;
    END;
/

CALL proc_cursor3();

-- Drop the stored procedure and the temporary table:
DROP PROCEDURE proc_cursor3;
DROP TABLE staffs_t1;
DROP TABLE sections_t1;
```

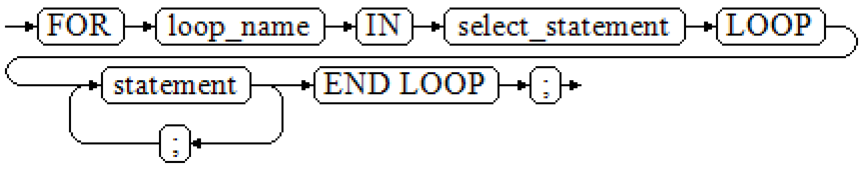
9.9.4 Cursor Loop

The use of cursors in **WHILE** and **LOOP** statements is called a cursor loop. Generally, **OPEN**, **FETCH**, and **CLOSE** statements are needed in cursor loop. The following describes a loop that is applicable to a static cursor loop without executing the four steps of a static cursor.

Syntax

Figure 9-33 shows the syntax diagram for the **FOR AS** loop.

Figure 9-33 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. The type and value of *loop_name* are the same as those of the query result of *select_statement*.
- The **%FOUND**, **%NOTFOUND**, and **%ROWCOUNT** attributes access the same internal variable in GaussDB(DWS). Transactions and anonymous blocks cannot be accessed by multiple cursors at the same time.

Examples

```
BEGIN
FOR ROW_TRANS IN
        SELECT first_name FROM staffs
    LOOP
        DBMS_OUTPUT.PUT_LINE (ROW_TRANS.first_name );
    END LOOP;
END;
/

-- Create a table:
CREATE TABLE integerTable1( A INTEGER) DISTRIBUTE BY hash(A);
CREATE TABLE integerTable2( B INTEGER) DISTRIBUTE BY hash(B);
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 value of C1%FOUND and C2%FOUND is FALSE.
   FETCH C2 INTO PI_B; ----  The value of C1%FOUND and C2%FOUND is TRUE.
-- Determine the cursor status:
   IF C1%FOUND THEN
       IF C2%FOUND THEN
          DBMS_OUTPUT.PUT_LINE('Dual cursor share paremeter.');
      END IF;
   END IF;
    CLOSE C1;-- Close the cursor.
   CLOSE C2;
END;
/

-- Drop the temporary table:
DROP TABLE integerTable1;
DROP TABLE integerTable2;
```

9.10 GaussDB(DWS) Stored Procedure Advanced Package

9.10.1 DBMS_LOB

Related Interfaces

Table 9-3 provides all interfaces supported by the **DBMS_LOB** package.

Table 9-3 DBMS_LOB

API	Description
DBMS_LOB.GETLENGTH	Obtains and returns the specified length of a LOB object.
DBMS_LOB.OPEN	Opens a LOB and returns a LOB descriptor.
DBMS_LOB.READ	Loads a part of LOB contents to BUFFER area according to the specified length and initial position offset.
DBMS_LOB.WRITE	Copies contents in BUFFER area to LOB according to the specified length and initial position offset.
DBMS_LOB.WRITEAPPEND	Copies contents in BUFFER area to the end part of LOB according to the specified length.
DBMS_LOB.COPY	Copies contents in BLOB to another BLOB according to the specified length and initial position offset.
DBMS_LOB.ERASE	Deletes contents in BLOB according to the specified length and initial position offset.
DBMS_LOB.CLOSE	Closes a LOB descriptor.
DBMS_LOB.INSTR	Returns the position of the Nth occurrence of a character string in LOB.
DBMS_LOB.COMPARE	Compares two LOBs or a certain part of two LOBs.

API	Description
DBMS_LOB.SUBSTR	Reads the substring of a LOB and returns the number of read bytes or the number of characters.
DBMS_LOB.TRIM	Truncates the LOB of a specified length. After the execution is complete, the length of the LOB is set to the length specified by the **newlen** parameter.
DBMS_LOB.CREATETEMPORARY	Creates a temporary BLOB or CLOB.
DBMS_LOB.APPEND	Adds the content of a LOB to another LOB.

- DBMS_LOB.GETLENGTH

  Specifies the length of a LOB type object obtained and returned by the stored procedure **GETLENGTH**.

  The function prototype of **DBMS_LOB.GETLENGTH** is:

```
DBMS_LOB.GETLENGTH (
lob_loc    IN   BLOB)
RETURN INTEGER;

DBMS_LOB.GETLENGTH (
lob_loc    IN   CLOB)
RETURN INTEGER;
```

Table 9-4 DBMS_LOB.GETLENGTH interface parameters

Parameter	Description
lob_loc	LOB type object whose length is to be obtained

- DBMS_LOB.OPEN

  A stored procedure opens a LOB and returns a LOB descriptor. This process is used only for compatibility.

  The function prototype of **DBMS_LOB.OPEN** is:

```
DBMS_LOB.LOB (
lob_loc INOUT BLOB,
open_mode IN BINARY_INTEGER);

DBMS_LOB.LOB (
lob_loc INOUT CLOB,
open_mode IN BINARY_INTEGER);
```

Table 9-5 DBMS_LOB.OPEN interface parameters

Parameter	Description
lob_loc	BLOB or CLOB descriptor that is opened
open_mode IN BINARY_INTEGER	Open mode (currently, DBMS_LOB.LOB_READWRITE is supported)

- DBMS_LOB.READ

  The stored procedure **READ** loads a part of LOB contents to BUFFER according to the specified length and initial position offset.

  The function prototype of **DBMS_LOB.READ** is:

```
DBMS_LOB.READ (
lob_loc     IN           BLOB,
amount      IN           INTEGER,
offset      IN           INTEGER,
buffer      OUT          RAW);

DBMS_LOB.READ (
lob_loc    IN            CLOB,
amount     IN OUT        INTEGER,
offset     IN            INTEGER,
buffer     OUT           VARCHAR2);
```

Table 9-6 DBMS_LOB.READ interface parameters

Parameter	Description
lob_loc	LOB type object to be loaded
amount	Load data length **NOTE** If the read length is negative, the error message "ERROR: argument 2 is null, invalid, or out of range." is displayed.
offset	Indicates where to start reading the LOB contents, that is, the offset bytes to initial position of LOB contents.
buffer	Target buffer to store the loaded LOB contents

- DBMS_LOB.WRITE

  The stored procedure **WRITE** copies contents in BUFFER to LOB variables according to the specified length and initial position offset.

  The function prototype of **DBMS_LOB.WRITE** is:

```
DBMS_LOB.WRITE (
lob_loc    IN OUT     BLOB,
amount     IN         INTEGER,
offset     IN         INTEGER,
buffer     IN         RAW);

DBMS_LOB.WRITE (
lob_loc   IN OUT      CLOB,
amount    IN          INTEGER,
offset    IN          INTEGER,
buffer    IN          VARCHAR2);
```

Table 9-7 DBMS_LOB.WRITE interface parameters

Parameter	Description
lob_loc	LOB type object to be written

Parameter	Description
amount	Write data length **NOTE** If the write data is shorter than 1 or longer than the contents to be written, an error is reported.
offset	Indicates where to start writing the LOB contents, that is, the offset bytes to initial position of LOB contents. **NOTE** If the offset is shorter than 1 or longer than the maximum length of LOB type contents, an error is reported.
buffer	Content to be written

- DBMS_LOB.WRITEAPPEND

  The stored procedure **WRITEAPPEND** copies contents in BUFFER to the end part of LOB according to the specified length.

  The function prototype of **DBMS_LOB.WRITEAPPEND** is:

```
DBMS_LOB.WRITEAPPEND (
lob_loc     IN OUT     BLOB,
amount      IN         INTEGER,
buffer      IN         RAW);

DBMS_LOB.WRITEAPPEND (
lob_loc     IN OUT     CLOB,
amount      IN         INTEGER,
buffer      IN         VARCHAR2);
```

Table 9-8 DBMS_LOB.WRITEAPPEND interface parameters

Parameter	Description
lob_loc	LOB type object to be written
amount	Write data length **NOTE** If the write data is shorter than 1 or longer than the contents to be written, an error is reported.
buffer	Content to be written

- DBMS_LOB.COPY

  The stored procedure **COPY** copies contents in BLOB to another BLOB according to the specified length and initial position offset.

  The function prototype of **DBMS_LOB.COPY** is:

```
DBMS_LOB.COPY (
dest_lob      IN OUT     BLOB,
src_lob       IN         BLOB,
amount        IN         INTEGER,
dest_offset   IN         INTEGER  DEFAULT 1,
src_offset    IN         INTEGER  DEFAULT 1);
```

Table 9-9 DBMS_LOB.COPY interface parameters

Parameter	Description
dest_lob	BLOB type object to be pasted
src_lob	BLOB type object to be copied
amount	Length of the copied data **NOTE** If the copied data is shorter than 1 or longer than the maximum length of BLOB type contents, an error is reported.
dest_offset	Indicates where to start pasting the BLOB contents, that is, the offset bytes to initial position of BLOB contents. **NOTE** If the offset is shorter than 1 or longer than the maximum length of BLOB type contents, an error is reported.
src_offset	Indicates where to start copying the BLOB contents, that is, the offset bytes to initial position of BLOB contents. **NOTE** If the offset is shorter than 1 or longer than the length of source BLOB, an error is reported.

- DBMS_LOB.ERASE

  The stored procedure **ERASE** deletes contents in BLOB according to the specified length and initial position offset.

  The function prototype of **DBMS_LOB.ERASE** is:

```
DBMS_LOB.ERASE (
lob_loc         IN OUT   BLOB,
amount           IN OUT   INTEGER,
offset          IN       INTEGER DEFAULT 1);
```

Table 9-10 DBMS_LOB.ERASE interface parameters

Parameter	Description
lob_loc	BLOB type object whose contents are to be deleted
amount	Length of contents to be deleted **NOTE** If the deleted data is shorter than 1 or longer than the maximum length of BLOB type contents, an error is reported.
offset	Indicates where to start deleting the BLOB contents, that is, the offset bytes to initial position of BLOB contents. **NOTE** If the offset is shorter than 1 or longer than the maximum length of BLOB type contents, an error is reported.

- DBMS_LOB.CLOSE

  The procedure **CLOSE** disables the enabled contents of LOB according to the specified length and initial position offset.

The function prototype of **DBMS_LOB.CLOSE** is:

```
DBMS_LOB.CLOSE(
src_lob       IN              BLOB);

DBMS_LOB.CLOSE (
src_lob      IN               CLOB);
```

Table 9-11 DBMS_LOB.CLOSE interface parameters

Parameter	Description
src_loc	LOB type object to be disabled

- DBMS_LOB.INSTR

  This function returns the Nth occurrence position in LOB. If invalid values are entered, **NULL** is returned. The invalid values include offset < 1 or offset > LOBMAXSIZE, nth < 1, and nth > LOBMAXSIZE.

  The function prototype of **DBMS_LOB.INSTR** is:

```
DBMS_LOB.INSTR (
lob_loc     IN     BLOB,
pattern     IN     RAW,
offset      IN     INTEGER := 1,
nth         IN     INTEGER := 1)
RETURN INTEGER;

DBMS_LOB.INSTR (
lob_loc    IN     CLOB,
pattern    IN     VARCHAR2 ,
offset     IN     INTEGER := 1,
nth        IN     INTEGER := 1)
RETURN INTEGER;
```

Table 9-12 DBMS_LOB.INSTR interface parameters

Parameter	Description
lob_loc	LOB descriptor to be searched for
pattern	Matched pattern. It is RAW for BLOB and TEXT for CLOB.
offset	For BLOB, the absolute offset is in the unit of byte. For CLOB, the offset is in the unit of character. The matching start position is 1.
nth	Number of pattern matching times. The minimum value is 1.

- DBMS_LOB.COMPARE

  This function compares two LOBs or a certain part of two LOBs.

  - If the two parts are equal, **0** is returned. Otherwise, a non-zero value is returned.
  - If the first CLOB is smaller than the second, **-1** is returned. If the first CLOB is larger than the second, **1** is returned.
  - If any of the **amount**, **offset_1**, and **offset_2** parameters is invalid, **NULL** is returned. The valid offset range is 1 to LOBMAXSIZE.

The function prototype of **DBMS_LOB.READ** is:

```
DBMS_LOB.COMPARE (
lob_1      IN      BLOB,
lob_2      IN      BLOB,
amount     IN      INTEGER := DBMS_LOB.LOBMAXSIZE,
offset_1   IN      INTEGER := 1,
offset_2   IN      INTEGER := 1)
RETURN INTEGER;

DBMS_LOB.COMPARE (
lob_1      IN      CLOB,
lob_2      IN      CLOB,
amount     IN      INTEGER := DBMS_LOB.LOBMAXSIZE,
offset_1   IN      INTEGER := 1,
offset_2   IN      INTEGER := 1)
RETURN INTEGER;
```

Table 9-13 DBMS_LOB.COMPARE interface parameters

Parameter	Description
lob_1	First LOB descriptor to be compared
lob_2	Second LOB descriptor to be compared
amount	Number of characters or bytes to be compared. The maximum value is DBMS_LOB.LOBMAXSIZE.
offset_1	Offset of the first LOB descriptor. The initial position is 1.
offset_2	Offset of the second LOB descriptor. The initial position is 1.

- DBMS_LOB.SUBSTR

  This function reads the substring of a LOB and returns the number of read bytes or the number of characters. If amount > 1, amount < 32767, offset < 1, or offset > LOBMAXSIZE, **NULL** is returned.

  The function prototype of **DBMS_LOB.SUBSTR** is:

```
DBMS_LOB.SUBSTR (
lob_loc     IN      BLOB,
amount      IN      INTEGER := 32767,
offset      IN      INTEGER := 1)
RETURN RAW;

DBMS_LOB.SUBSTR (
lob_loc    IN      CLOB,
amount     IN      INTEGER := 32767,
offset     IN      INTEGER := 1)
RETURN VARCHAR2;
```

Table 9-14 DBMS_LOB.SUBSTR interface parameters

Parameter	Description
lob_loc	LOB descriptor of the substring to be read. For BLOB, the return value is the number of read bytes. For CLOB, the return value is the number of characters.
offset	Number of bytes or characters to be read.

Parameter	Description
buffer	Number of characters or bytes offset from the start position.

- DBMS_LOB.TRIM

  This stored procedure truncates the LOB of a specified length. After this stored procedure is executed, the length of the LOB is set to the length specified by the **newlen** parameter. If an empty LOB is truncated, no execution result is displayed. If the specified length is longer than the length of LOB, an exception occurs.

  The function prototype of **DBMS_LOB.TRIM** is:

```
DBMS_LOB.TRIM (
lob_loc     IN OUT     BLOB,
newlen      IN         INTEGER);

DBMS_LOB.TRIM (
lob_loc    IN          OUT CLOB,
newlen     IN          INTEGER);
```

Table 9-15 DBMS_LOB.TRIM interface parameters

Parameter	Description
lob_loc	BLOB type object to be read
newlen	After truncation, the new LOB length for BLOB is in the unit of byte and that for CLOB is in the unit of character.

- DBMS_LOB.CREATETEMPORARY

  This stored procedure creates a temporary BLOB or CLOB and is used only for syntax compatibility.

  The function prototype of **DBMS_LOB.CREATETEMPORARY** is:

```
DBMS_LOB.CREATETEMPORARY (
lob_loc    IN OUT      BLOB,
cache      IN          BOOLEAN,
dur        IN          INTEGER);

DBMS_LOB.CREATETEMPORARY (
lob_loc    IN OUT     CLOB,
cache      IN         BOOLEAN,
dur        IN         INTEGER);
```

Table 9-16 DBMS_LOB.CREATETEMPORARY interface parameters

Parameter	Description
lob_loc	LOB descriptor
cache	This parameter is used only for syntax compatibility.
dur	This parameter is used only for syntax compatibility.

- DBMS_LOB.APPEND

  The stored procedure **READ** loads a part of BLOB contents to BUFFER according to the specified length and initial position offset.

  The function prototype of **DBMS_LOB.APPEND** is:

```
DBMS_LOB.APPEND (
dest_lob    IN OUT       BLOB,
src_lob     IN           BLOB);

DBMS_LOB.APPEND (
dest_lob    IN OUT       CLOB,
src_lob     IN           CLOB);
```

Table 9-17 DBMS_LOB.APPEND interface parameters

Parameter	Description
dest_lob	LOB descriptor to be written
src_lob	LOB descriptor to be read

Examples

```
-- Obtain the length of the character string.
SELECT DBMS_LOB.GETLENGTH('12345678');

DECLARE
myraw  RAW(100);
amount INTEGER :=2;
buffer INTEGER :=1;
begin
DBMS_LOB.READ('123456789012345',amount,buffer,myraw);
dbms_output.put_line(myraw);
end;
/

CREATE TABLE blob_Table (t1 blob) DISTRIBUTE BY REPLICATION;
CREATE TABLE blob_Table_bak (t2 blob) DISTRIBUTE BY REPLICATION;
INSERT INTO blob_Table VALUES('abcdef');
INSERT INTO blob_Table_bak VALUES('22222');

DECLARE
str varchar2(100) := 'abcdef';
source raw(100);
dest blob;
copyto blob;
amount int;
PSV_SQL varchar2(100);
PSV_SQL1 varchar2(100);
a int :=1;
len int;
BEGIN
source := utl_raw.cast_to_raw(str);
amount := utl_raw.length(source);

PSV_SQL :='select * from blob_Table for update';
PSV_SQL1 := 'select * from blob_Table_bak for update';

EXECUTE IMMEDIATE PSV_SQL into dest;
EXECUTE IMMEDIATE PSV_SQL1 into copyto;

DBMS_LOB.WRITE(dest, amount, 1, source);
DBMS_LOB.WRITEAPPEND(dest, amount, source);
DBMS_LOB.ERASE(dest, a, 1);
DBMS_OUTPUT.PUT_LINE(a);
DBMS_LOB.COPY(copyto, dest, amount, 10, 1);
DBMS_LOB.CLOSE(dest);
RETURN;
END;
/

--Delete the table.
DROP TABLE blob_Table;
DROP TABLE blob_Table_bak;
```

9.10.2 DBMS_RANDOM

Related Interfaces

Table 9-18 provides all interfaces supported by the **DBMS_RANDOM** package.

Table 9-18 DBMS_RANDOM interface parameters

Interface	Description
DBMS_RANDOM.SEED	Sets a seed for a random number.
DBMS_RANDOM.VALUE	Generates a random number between a specified low and a specified high.

- DBMS_RANDOM.SEED

  The stored procedure SEED is used to set a seed for a random number. The DBMS_RANDOM.SEED function prototype is:

  ```
  DBMS_RANDOM.SEED (seed IN INTEGER);
  ```

  Table 9-19 DBMS_RANDOM.SEED interface parameters

Parameter	Description
seed	Generates a seed for a random number.

- DBMS_RANDOM.VALUE

  The stored procedure VALUE generates a random number between a specified low and a specified high. The DBMS_RANDOM.VALUE function prototype is:

  ```
  DBMS_RANDOM.VALUE(
  low  IN  NUMBER,
  high IN  NUMBER)
  RETURN NUMBER;
  ```

  Table 9-20 DBMS_RANDOM.VALUE interface parameters

Parameter	Description
low	Sets the low bound for a random number. The generated random number is greater than or equal to the low.

Parameter	Description
high	Sets the high bound for a random number. The generated random number is less than the high.

NOTE

The only requirement is that the parameter type is **NUMERIC** regardless of the right and left bound values.

Example

Generate a random number between 0 and 1.

```
SELECT DBMS_RANDOM.VALUE(0,1);
```

Specify the low and high parameters to an integer within the specified range and intercept smaller values from the result. (The maximum value cannot be a possible value.) Therefore, use the following code for an integer between 0 and 99:

```
SELECT TRUNC(DBMS_RANDOM.VALUE(0,100));
```

9.10.3 DBMS_OUTPUT

Related Interfaces

Table 9-21 provides all interfaces supported by the **DBMS_OUTPUT** package.

Table 9-21 DBMS_OUTPUT

API	Description
DBMS_OUTPUT.PUT_LINE	Outputs the specified text. The text length cannot exceed 32,767 bytes.
DBMS_OUTPUT.PUT	Outputs the specified text to the front of the specified text without adding a line break. The text length cannot exceed 32,767 bytes.
DBMS_OUTPUT.ENABLE	Sets the buffer area size. If this interface is not specified, the maximum buffer size is 20,000 bytes and the minimum buffer size is 2000 bytes. If the specified buffer size is less than 2000 bytes, the default minimum buffer size is applied.

- DBMS_OUTPUT.PUT_LINE

The PUT_LINE procedure writes a row of text carrying a line end symbol in the buffer. The DBMS_OUTPUT.PUT_LINE function prototype is:

```
DBMS_OUTPUT.PUT_LINE (
item IN VARCHAR2);
```

Table 9-22 DBMS_OUTPUT.PUT_LINE interface parameters

Parameter	Description
item	Specifies the text that was written to the buffer.

- DBMS_OUTPUT.PUT

The stored procedure **PUT** outputs the specified text to the front of the specified text without adding a linefeed. The DBMS_OUTPUT.PUT function prototype is:

```
DBMS_OUTPUT.PUT (
item IN VARCHAR2);
```

Table 9-23 DBMS_OUTPUT.PUT interface parameters

Parameter	Description
item	Specifies the text that was written to the specified text.

- DBMS_OUTPUT.ENABLE

The stored procedure **ENABLE** sets the output buffer size. If the size is not specified, it contains a maximum of 20,000 bytes. The DBMS_OUTPUT.ENABLE function prototype is:

```
DBMS_OUTPUT.ENABLE (
buf IN INTEGER);
```

Table 9-24 DBMS_OUTPUT.ENABLE interface parameters

Parameter	Description
buf	Sets the buffer area size.

Examples

```
BEGIN
    DBMS_OUTPUT.ENABLE(50);
    DBMS_OUTPUT.PUT ('hello, ');
    DBMS_OUTPUT.PUT_LINE('database!');-- Displaying "hello, database!"
END;
/
```

9.10.4 UTL_RAW

Related Interfaces

Table 9-25 provides all interfaces supported by the **UTL_RAW** package.

Table 9-25 UTL_RAW

API	Description
UTL_RAW.CAST_FROM_BINARY_INTEGER	Converts an INTEGER type value to a binary representation (RAW type).
UTL_RAW.CAST_TO_BINARY_INTEGER	Converts a binary representation (RAW type) to an INTEGER type value.
UTL_RAW.LENGTH	Obtains the length of the RAW type object.
UTL_RAW.CAST_TO_RAW	Converts a VARCHAR2 type value to a binary expression (RAW type).

NOTICE

The external representation of the RAW type data is hexadecimal and its internal storage form is binary. For example, the representation of the **RAW** type data **11001011** is 'CB'. The input of the actual type conversion is 'CB'.

- UTL_RAW.CAST_FROM_BINARY_INTEGER

  The stored procedure **CAST_FROM_BINARY_INTEGER** converts an **INTEGER** type value to a binary representation (**RAW** type).

  The **UTL_RAW.CAST_FROM_BINARY_INTEGER** function prototype is:

```
UTL_RAW.CAST_FROM_BINARY_INTEGER (
n           IN  INTEGER,
endianess   IN  INTEGER)
RETURN RAW;
```

Table 9-26 UTL_RAW.CAST_FROM_BINARY_INTEGER interface parameters

Parameter	Description
n	Specifies the INTEGER type value to be converted to the RAW type.
endianess	Specifies the **INTEGER** type value **1** or **2** of the byte sequence. (**1** indicates **BIG_ENDIAN** and **2** indicates **LITTLE-ENDIAN**.)

- UTL_RAW.CAST_TO_BINARY_INTEGER

  The stored procedure CAST_TO_BINARY_INTEGER converts an INTEGER type value in a binary representation (RAW type) to the INTEGER type.

  The UTL_RAW.CAST_TO_BINARY_INTEGER function prototype is:

```
UTL_RAW.CAST_TO_BINARY_INTEGER (
r          IN  RAW,
endianess  IN  INTEGER)
RETURN BINARY_INTEGER;
```

Table 9-27 UTL_RAW.CAST_TO_BINARY_INTEGER interface parameters

Parameter	Description
r	Specifies an INTEGER type value in a binary representation (RAW type).
endianess	Specifies the **INTEGER** type value **1** or **2** of the byte sequence. (**1** indicates **BIG_ENDIAN** and **2** indicates **LITTLE-ENDIAN**.)

- UTL_RAW.LENGTH

  The stored procedure LENGTH returns the length of a RAW type object.

  The UTL_RAW.LENGTH function prototype is:

```
UTL_RAW.LENGTH(
r      IN RAW)
RETURN INTEGER;
```

Table 9-28 UTL_RAW.LENGTH interface parameters

Parameter	Description
r	Specifies a RAW type object.

- UTL_RAW.CAST_TO_RAW

  The stored procedure CAST_TO_RAW converts a VARCHAR2 type object to the RAW type.

  The UTL_RAW.CAST_TO_RAW function prototype is:

```
UTL_RAW.CAST_TO_RAW(
c      IN VARCHAR2)
RETURN RAW;
```

Table 9-29 UTL_RAW.CAST_TO_RAW interface parameters

Parameter	Description
c	Specifies a VARCHAR2 type object to be converted.

Example

Perform operations on RAW data in a stored procedure.

```
--
CREATE OR REPLACE PROCEDURE proc_raw
AS
str varchar2(100) := 'abcdef';
source raw(100);
amount integer;
BEGIN
source := utl_raw.cast_to_raw(str);--Convert the type.
amount := utl_raw.length(source);--Obtain the length.
dbms_output.put_line(amount);
END;
/
```

Call the stored procedure.

```
CALL proc_raw();
```

9.10.5 DBMS_JOB

Related Interfaces

Table 9-30 lists all interfaces supported by the **DBMS_JOB** package.

Table 9-30 DBMS_JOB

Interface	Description
DBMS_JOB.SUBMIT	Submits a job to the job queue. The job number is automatically generated by the system.
DBMS_JOB.ISUBMIT	Submits a job to the job queue. The job number is specified by the user.
DBMS_JOB.REMOVE	Removes a job from the job queue by job number.
DBMS_JOB.BROKEN	Disables or enables job execution.
DBMS_JOB.CHANGE	Modifies user-definable attributes of a job, including the job description, next execution time, and execution interval.
DBMS_JOB.WHAT	Modifies the job description of a job.
DBMS_JOB.NEXT_DATE	Modifies the next execution time of a job.
DBMS_JOB.INTERVAL	Modifies the execution interval of a job.
DBMS_JOB.CHANGE_OWNER	Modifies the owner of a job.

- DBMS_JOB.SUBMIT

  The stored procedure **SUBMIT** submits a job provided by the system.

  A prototype of the DBMS_JOB.SUBMIT function is as follows:

  ```
  DMBS_JOB.SUBMIT(
  what         IN   TEXT,
  next_date    IN   TIMESTAMP DEFAULT sysdate,
  job_interval IN   TEXT  DEFAULT 'null',
  job          OUT  INTEGER);
  ```

  NOTE

  When a job is created (using DBMS_JOB), the system binds the current database and the username to the job by default. This function can be invoked by using **call** or **select**. If you invoke this function by using **select**, there is no need to specify output parameters. To invoke this function within a stored procedure, use **perform**.

Table 9-31 DBMS_JOB.SUBMIT interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
what	text	IN	No	SQL statement to be executed. One or multiple DMLs, anonymous blocks, and SQL statements that invoke stored procedures, or all three combined are supported.
next_date	timestamp	IN	No	Specifies the next time the job will be executed. The default value is the current system time (sysdate). If the specified time has past, the job is executed at the time it is submitted.
interval	text	IN	Yes	Calculates the next time to execute the job. It can be an interval expression, or sysdate followed by a numeric value, for example, **sysdate+1.0/24**. If this parameter is left blank or set to **null**, the job will be executed only once, and the job status will change to **'d'** afterward.
job	integer	OUT	No	Specifies the job number. The value ranges from 1 to 32767. When **dbms.submit** is invoked using **select**, this parameter can be skipped.

For example:

```
select DBMS_JOB.SUBMIT('call pro_xxx();', to_date('20180101','yyyymmdd'),'sysdate+1');

select DBMS_JOB.SUBMIT('call pro_xxx();', to_date('20180101','yyyymmdd'),'sysdate+1.0/24');

CALL DBMS_JOB.SUBMIT('INSERT INTO T_JOB  VALUES(1);  call pro_1(); call pro_2();',
add_months(to_date('201701','yyyymm'),1), 'date_trunc(''day'',SYSDATE) + 1 +(8*60+30.0)/
(24*60)' ,:jobid);
```

- DBMS_JOB.ISUBMIT

  ISUBMIT has the same syntax function as **SUBMIT**, but the first parameter of **ISUBMIT** is an input parameter, that is, a specified job number. In contrast, that last parameter of **SUBMIT** is an output parameter, indicating the job number automatically generated by the system.

  For example:

  ```
  CALL dbms_job.isubmit(101, 'insert_msg_statistic1;', sysdate, 'sysdate+3.0/24');
  ```

- DBMS_JOB.REMOVE

  The stored procedure **REMOVE** deletes a specified job.

  A prototype of the DBMS_JOB.REMOVE function is as follows:

  ```
  REMOVE(job  IN  INTEGER);
  ```

Table 9-32 DBMS_JOB.REMOVE interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
job	integer	IN	No	Specifies the job number.

For example:

```
CALL dbms_job.remove(101);
```

- DBMS_JOB.BROKEN

  The stored procedure **BROKEN** sets the broken flag of a job.

  A prototype of the DBMS_JOB.BROKEN function is as follows:

```
DMBS_JOB.BROKEN(
job          IN   INTEGER,
broken       IN   BOOLEAN,
next_date    IN   TIMESTAMP  DEFAULT  sysdate);
```

Table 9-33 DBMS_JOB.BROKEN interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
job	integer	IN	No	Specifies the job number.
broken	boolean	IN	No	Specifies the status flag, **true** for broken and **false** for not broken. Setting this parameter to **true** or **false** updates the current job. If the parameter is left blank, the job status remains unchanged.
next_date	timestamp	IN	Yes	Specifies the next execution time. The default is the current system time. If **broken** is set to **true**, **next_date** is updated to **'4000-1-1'**. If **broken** is **false** and **next_date** is not empty, **next_date** is updated for the job. If **next_date** is empty, it will not be updated. This parameter can be omitted, and its default value will be used in this case.

For example:

```
CALL dbms_job.broken(101, true);
CALL dbms_job.broken(101, false, sysdate);
```

- DBMS_JOB.CHANGE

  The stored procedure **CHANGE** modifies user-definable attributes of a job, including the job content, next-execution time, and execution interval.

  A prototype of the DBMS_JOB.CHANGE function is as follows:

```
DMBS_JOB.CHANGE(
job          IN   INTEGER,
what          IN   TEXT,
next_date    IN   TIMESTAMP,
interval     IN   TEXT);
```

Table 9-34 DBMS_JOB.CHANGE interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
job	integer	IN	No	Specifies the job number.
what	text	IN	Yes	Specifies the name of the stored procedure or SQL statement block that is executed. If this parameter is left blank, the system does not update the **what** parameter for the specified job. Otherwise, the system updates the **what** parameter for the specified job.
next_date	timestamp	IN	Yes	Specifies the next execution time. If this parameter is left blank, the system does not update the **next_date** parameter for the specified job. Otherwise, the system updates the **next_date** parameter for the specified job.
interval	text	IN	Yes	Specifies the time expression for calculating the next time the job will be executed. If this parameter is left blank, the system does not update the **interval** parameter for the specified job. Otherwise, the system updates the **interval** parameter for the specified job after necessary validity check. If this parameter is set to **null**, the job will be executed only once, and the job status will change to **'d'** afterward.

For example:

```
CALL dbms_job.change(101, 'call userproc();', sysdate, 'sysdate + 1.0/1440');
CALL dbms_job.change(101, 'insert into tbl_a values(sysdate);', sysdate, 'sysdate + 1.0/1440');
```

- DBMS_JOB.WHAT

  The stored procedure **WHAT** modifies the procedures to be executed by a specified job.

  A prototype of the DBMS_JOB.WHAT function is as follows:

```
DMBS_JOB.WHAT(
job              IN      INTEGER,
what             IN      TEXT);
```

Table 9-35 DBMS_JOB.WHAT interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
job	integer	IN	No	Specifies the job number.
what	text	IN	No	Specifies the name of the stored procedure or SQL statement block that is executed.

NOTE

- If the value specified by the **what** parameter is one or multiple executable SQL statements, program blocks, or stored procedures, this procedure can be executed successfully; otherwise, it will fail to be executed.
- If the **what** parameter is a simple statement such as insert and update, a schema name must be added in front of the table name.

For example:

```
CALL dbms_job.what(101, 'call userproc();');
CALL dbms_job.what(101, 'insert into tbl_a values(sysdate);');
```

- DBMS_JOB.NEXT_DATE

  The stored procedure **NEXT_DATE** modifies the next-execution time attribute of a job.

  A prototype of the DBMS_JOB.NEXT_DATE function is as follows:

```
DMBS_JOB.NEXT_DATE(
job          IN    INTEGER,
next_date    IN    TIMESTAMP);
```

Table 9-36 DBMS_JOB.NEXT_DATE interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
job	integer	IN	No	Specifies the job number.

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
next_date	timestamp	IN	No	Specifies the next execution time.

NOTE

If the specified **next_date** value is earlier than the current date, the job is executed once immediately.

For example:

```
CALL dbms_job.next_date(101, sysdate);
```

- DBMS_JOB.INTERVAL

  The stored procedure **INTERVAL** modifies the execution interval attribute of a job.

  A prototype of the DBMS_JOB.INTERVAL function is as follows:

```
DMBS_JOB.INTERVAL(
job             IN   INTEGER,
interval         IN   TEXT);
```

Table 9-37 DBMS_JOB.INTERVAL interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
job	integer	IN	No	Specifies the job number.
interval	text	IN	Yes	Specifies the time expression for calculating the next time the job will be executed. If this parameter is left blank or set to **null**, the job will be executed only once, and the job status will change to **'d'** afterward. **interval** must be a valid time or interval type.

For example:

```
CALL dbms_job.interval(101, 'sysdate + 1.0/1440');
```

NOTE

For a job that is currently running (that is, **job_status** is **'r'**), it is not allowed to use **remove**, **change**, **next_date**, **what**, or **interval** to delete or modify job parameters.

- DBMS_JOB.CHANGE_OWNER

  The stored procedure **CHANGE_OWNER** modifies the owner of a job.

  A prototype of the DBMS_JOB.CHANGE_OWNER function is as follows:

```
DMBS_JOB.CHANGE_OWNER(
job            IN     INTEGER,
new_owner       IN     NAME);
```

Table 9-38 DBMS_JOB.CHANGE_OWNER interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
job	integer	IN	No	Specifies the job number.
new_owner	name	IN	No	Specifies the new username.

For example:

```
CALL dbms_job.change_owner(101, 'alice');
```

Constraints

1. After a new job is created, this job belongs to the current coordinator only, that is, this job can be scheduled and executed only on the current coordinator. Other coordinators will not schedule or execute this job. All coordinators can query, modify, and delete jobs created on other CNs.
2. Create, update, and delete jobs only using the procedures provided by the DBMS_JOB package. These procedures synchronize job information between different CNs and associate primary keys between the **pg_jobs** tables. If you use DML statements to add, delete, or modify records in the **pg_jobs** table, job information will become inconsistent between CNs and system tables may fail to be associated, compromising internal job management.
3. Each user-created task is bound to a CN. If the automatic migration function is not enabled, task statuses cannot be updated in real time when the CN is faulty during task execution. When a CN fails, all jobs on this CN cannot be scheduled or executed until the CN is restored manually. Enable the automatic migration function on CNs, so that jobs on the faulty CN will be migrated to other CNs for scheduling.
4. For each job, the hosting CN updates the real-time job information (including the job status, last execution start time, last execution end time, next execution start time, the number of execution failures if any) to the **pg_jobs** table, and synchronizes the information to other CNs, ensuring consistent job information between different CNs. In the case of CN failures, job information synchronization is reattempted by the hosting CNs, which increases job execution time. Although job information fails to be synchronized between CNs, job information can still be properly updated in the **pg_jobs** table on the hosting CNs, and jobs can be executed successfully. After a CN recovers, job information such as job execution time and status in its **pg_jobs** table may be

incorrect and will be updated only after the jobs are executed again on related CNs.

5. For each job, a thread is established to execute it. If multiple jobs are triggered concurrently as scheduled, the system will need some time to start the required threads, resulting in a latency of 0.1 ms in job execution.
6. The length of the SQL statement to be executed in a job is limited. The maximum length is 8 KB.

9.10.6 DBMS_SQL

Related Interfaces

Table 9-39 lists interfaces supported by the **DBMS_SQL** package.

Table 9-39 DBMS_SQL

API	Description
DBMS_SQL.OPEN_CURSOR	Opens a cursor.
DBMS_SQL.CLOSE_CURSOR	Closes an open cursor.
DBMS_SQL.PARSE	Transmits a group of SQL statements to a cursor. Currently, only the **SELECT** statement is supported.
DBMS_SQL.EXECUTE	Performs a set of dynamically defined operations on the cursor.
DBMS_SQL.FETCHE_ROWS	Reads a row of cursor data.
DBMS_SQL.DEFINE_COLUMN	Dynamically defines a column.
DBMS_SQL.DEFINE_COLUMN_CHAR	Dynamically defines a column of the CHAR type.
DBMS_SQL.DEFINE_COLUMN_INT	Dynamically defines a column of the INT type.
DBMS_SQL.DEFINE_COLUMN_LONG	Dynamically defines a column of the LONG type.
DBMS_SQL.DEFINE_COLUMN_RAW	Dynamically defines a column of the RAW type.
DBMS_SQL.DEFINE_COLUMN_TEXT	Dynamically defines a column of the TEXT type.
DBMS_SQL.DEFINE_COLUMN_UNKNOWN	Dynamically defines a column of an unknown type.
DBMS_SQL.COLUMN_VALUE	Reads a dynamically defined column value.

API	Description
DBMS_SQL.COLUMN_VALUE_CHAR	Reads a dynamically defined column value of the CHAR type.
DBMS_SQL.COLUMN_VALUE_INT	Reads a dynamically defined column value of the INT type.
DBMS_SQL.COLUMN_VALUE_LONG	Reads a dynamically defined column value of the LONG type.
DBMS_SQL.COLUMN_VALUE_RAW	Reads a dynamically defined column value of the RAW type.
DBMS_SQL.COLUMN_VALUE_TEXT	Reads a dynamically defined column value of the TEXT type.
DBMS_SQL.COLUMN_VALUE_UNKNOWN	Reads a dynamically defined column value of an unknown type.
DBMS_SQL.IS_OPEN	Checks whether a cursor is opened.

NOTE

- You are advised to use **dbms_sql.define_column** and **dbms_sql.column_value** to define columns.
- If the size of the result set is greater than the value of **work_mem**, the result set will be flushed to disk. The value of **work_mem** must be no greater than 512 MB.

- DBMS_SQL.OPEN_CURSOR

  This function opens a cursor and is the prerequisite for the subsequent dbms_sql operations. This function does not transfer any parameter. It automatically generates cursor IDs in an ascending order and returns values to integer variables.

  The function prototype of **DBMS_SQL.OPEN_CURSOR** is:

```
DBMS_SQL.OPEN_CURSOR (
)
RETURN INTEGER;
```

- DBMS_SQL.CLOSE_CURSOR

  This function closes a cursor. It is the end of each dbms_sql operation. If this function is not invoked when the stored procedure ends, the memory is still occupied by the cursor. Therefore, remember to close a cursor when you do not need to use it. If an exception occurs, the stored procedure exits but the cursor is not closed. Therefore, you are advised to include this interface in the exception handling of the stored procedure.

  The function prototype of **DBMS_SQL.CLOSE_CURSOR** is:

```
DBMS_SQL.CLOSE_CURSOR (
cursorid     IN INTEGER
)
RETURN INTEGER;
```

Table 9-40 DBMS_SQL.CLOSE_CURSOR interface parameters

Parameter Name	Description
cursorid	ID of the cursor to be closed

- DBMS_SQL.PARSE

  This function parses the query statement of a given cursor. The input query statement is executed immediately. Currently, only the **SELECT** query statement can be parsed. The statement parameters can be transferred only through the TEXT type. The length cannot exceed 1 GB.

  The function prototype of **DBMS_SQL.PARSE** is:

```
DBMS_SQL.PARSE (
cursorid     IN INTEGER,
query_string IN TEXT,
label        IN INTEGER
)
RETURN BOOLEAN;
```

Table 9-41 DBMS_SQL.PARSE interface parameters

Parameter Name	Description
cursorid	ID of the cursor whose query statement is parsed
query_string	Query statements to be parsed
language_flag	Version language number. Currently, only **1** is supported.

- DBMS_SQL.EXECUTE

  This function executes a given cursor. This function receives a cursor ID. The obtained data after is used for subsequent operations. Currently, only the **SELECT** query statement can be executed.

  The function prototype of **DBMS_SQL.EXECUTE** is:

```
DBMS_SQL.EXECUTE(
cursorid     IN INTEGER,
)
RETURN INTEGER;
```

Table 9-42 DBMS_SQL.EXECUTE interface parameters

Parameter Name	Description
cursorid	ID of the cursor whose query statement is parsed

- DBMS_SQL.FETCHE_ROWS

  This function returns the number of data rows that meet query conditions. Each time the interface is executed, the system obtains a set of new rows until all data is read.

  The function prototype of **DBMS_SQL.FETCHE_ROWS** is:

```
DBMS_SQL.FETCHE_ROWS(
cursorid     IN INTEGER,
)
RETURN INTEGER;
```

Table 9-43 DBMS_SQL.FETCH_ROWS interface parameters

Parameter Name	Description
curosorid	ID of the cursor to be executed

- DBMS_SQL.DEFINE_COLUMN

  This function defines columns returned from a given cursor and can be used only for the cursors defined by **SELECT**. The defined columns are identified by the relative positions in the query list. The data type of the input variable determines the column type.

  The function prototype of **DBMS_SQL.DEFINE_COLUMN** is:

```
DBMS_SQL.DEFINE_COLUMN(
cursorid     IN INTEGER,
position     IN INTEGER,
column_ref   IN ANYELEMENT,
column_size     IN INTEGER default 1024
)
RETURN INTEGER;
```

Table 9-44 DBMS_SQL.DEFINE_COLUMN interface parameters

Parameter Name	Description
cursorid	ID of the cursor to be executed
position	Position of a dynamically defined column in the query
column_ref	Variable of any type. You can select an appropriate interface to dynamically define columns based on variable types.
column_size	Length of a defined column

- DBMS_SQL.DEFINE_COLUMN_CHAR

  This function defines columns of the CHAR type returned from a given cursor and can be used only for the cursors defined by **SELECT**. The defined columns are identified by the relative positions in the query list. The data type of the input variable determines the column type.

  The function prototype of **DBMS_SQL.DEFINE_COLUMN_CHAR** is:

```
DBMS_SQL.DEFINE_COLUMN_CHAR(
cursorid     IN INTEGER,
position     IN INTEGER,
column       IN TEXT,
column_size     IN INTEGER
)
RETURN INTEGER;
```

Table 9-45 DBMS_SQL.DEFINE_COLUMN_CHAR interface parameters

Parameter Name	Description
cursorid	ID of the cursor to be executed
position	Position of a dynamically defined column in the query
column	Parameter to be defined
column_size	Length of a dynamically defined column

- DBMS_SQL.DEFINE_COLUMN_INT

  This function defines columns of the INT type returned from a given cursor and can be used only for the cursors defined by **SELECT**. The defined columns are identified by the relative positions in the query list. The data type of the input variable determines the column type.

  The function prototype of **DBMS_SQL.DEFINE_COLUMN_INT** is:

```
DBMS_SQL.DEFINE_COLUMN_INT(
cursorid     IN INTEGER,
position     IN INTEGER
)
RETURN INTEGER;
```

Table 9-46 DBMS_SQL.DEFINE_COLUMN_INT interface parameters

Parameter Name	Description
cursorid	ID of the cursor to be executed
position	Position of a dynamically defined column in the query

- DBMS_SQL.DEFINE_COLUMN_LONG

  This function defines columns of a long type (not LONG) returned from a given cursor and can be used only for the cursors defined by **SELECT**. The defined columns are identified by the relative positions in the query list. The data type of the input variable determines the column type. The maximum size of a long column is 1 GB.

  The function prototype of **DBMS_SQL.DEFINE_COLUMN_LONG** is:

```
DBMS_SQL.DEFINE_COLUMN_LONG(
cursorid     IN INTEGER,
position     IN INTEGER
)
RETURN INTEGER;
```

Table 9-47 DBMS_SQL.DEFINE_COLUMN_LONG interface parameters

Parameter Name	Description
cursorid	ID of the cursor to be executed

Parameter Name	Description
position	Position of a dynamically defined column in the query

- DBMS_SQL.DEFINE_COLUMN_RAW

  This function defines columns of the RAW type returned from a given cursor and can be used only for the cursors defined by **SELECT**. The defined columns are identified by the relative positions in the query list. The data type of the input variable determines the column type.

  The function prototype of **DBMS_SQL.DEFINE_COLUMN_RAW** is:

```
DBMS_SQL.DEFINE_COLUMN_RAW(
cursorid     IN INTEGER,
position     IN INTEGER,
column       IN BYTEA,
column_size     IN INTEGER
)
RETURN INTEGER;
```

Table 9-48 DBMS_SQL.DEFINE_COLUMN_RAW interface parameters

Parameter Name	Description
cursorid	ID of the cursor to be executed
position	Position of a dynamically defined column in the query
column	Parameter of the RAW type
column_size	Column length

- DBMS_SQL.DEFINE_COLUMN_TEXT

  This function defines columns of the TEXT type returned from a given cursor and can be used only for the cursors defined by **SELECT**. The defined columns are identified by the relative positions in the query list. The data type of the input variable determines the column type.

  The function prototype of **DBMS_SQL.DEFINE_COLUMN_TEXT** is:

```
DBMS_SQL.DEFINE_COLUMN_CHAR(
cursorid     IN INTEGER,
position     IN INTEGER,
max_size     IN INTEGER
)
RETURN INTEGER;
```

Table 9-49 DBMS_SQL.DEFINE_COLUMN_TEXT interface parameters

Parameter Name	Description
cursorid	ID of the cursor to be executed
position	Position of a dynamically defined column in the query

Parameter Name	Description
max_size	Maximum length of the defined TEXT type

- DBMS_SQL.DEFINE_COLUMN_UNKNOWN

  This function processes columns of unknown data types returned from a given cursor and is used only for the system to report an error and exist when the type cannot be identified.

  The function prototype of **DBMS_SQL.DEFINE_COLUMN_UNKNOWN** is:

```
DBMS_SQL.DEFINE_COLUMN_CHAR(
cursorid     IN INTEGER,
position     IN INTEGER,
column       IN TEXT
)
RETURN INTEGER;
```

Table 9-50 DBMS_SQL.DEFINE_COLUMN_UNKNOWN interface parameters

Parameter Name	Description
cursorid	ID of the cursor to be executed
position	Position of a dynamically defined column in the query
column	Dynamically defined parameter

- DBMS_SQL.COLUMN_VALUE

  This function returns the cursor element value specified by a cursor and accesses the data obtained by DBMS_SQL.FETCH_ROWS.

  The function prototype of **DBMS_SQL.COLUMN_VALUE** is:

```
DBMS_SQL.COLUMN_VALUE(
cursorid                 IN    INTEGER,
position                 IN    INTEGER,
column_value             INOUT ANYELEMENT
)
RETURN ANYELEMENT;
```

Table 9-51 DBMS_SQL.COLUMN_VALUE interface parameters

Parameter Name	Description
cursorid	ID of the cursor to be executed
position	Position of a dynamically defined column in the query
column_value	Return value of a defined column

- DBMS_SQL.COLUMN_VALUE_CHAR

  This function returns the value of the CHAR type in a specified position of a cursor and accesses the data obtained by DBMS_SQL.FETCH_ROWS.

The function prototype of **DBMS_SQL.COLUMN_VALUE_CHAR** is:

```
DBMS_SQL.COLUMN_VALUE_CHAR(
cursorid                 IN    INTEGER,
position                 IN    INTEGER,
column_value             INOUT CHARACTER,
err_num                  INOUT NUMERIC default 0,
actual_length            INOUT INTEGER default 1024
)
RETURN RECORD;
```

Table 9-52 DBMS_SQL.COLUMN_VALUE_CHAR interface parameters

Parameter Name	Description
cursorid	ID of the cursor to be executed
position	Position of a dynamically defined column in the query
column_value	Return value
err_num	Error No. It is an output parameter and the argument must be a variable. Currently, the output value is **–1** regardless of the argument.
actual_length	Length of a return value

- DBMS_SQL.COLUMN_VALUE_INT

  This function returns the value of the INT type in a specified position of a cursor and accesses the data obtained by DBMS_SQL.FETCH_ROWS. The function prototype of **DBMS_SQL.COLUMN_VALUE_INT** is:

```
DBMS_SQL.COLUMN_VALUE_INT(
cursorid                 IN    INTEGER,
position                 IN    INTEGER
)
RETURN INTEGER;
```

Table 9-53 DBMS_SQL.COLUMN_VALUE_INT interface parameters

Parameter Name	Description
cursorid	ID of the cursor to be executed
position	Position of a dynamically defined column in the query

- DBMS_SQL.COLUMN_VALUE_LONG

  This function returns the value of a long type (not LONG or BIGINT) in a specified position of a cursor and accesses the data obtained by DBMS_SQL.FETCH_ROWS.

  The function prototype of **DBMS_SQL.COLUMN_VALUE_LONG** is:

```
DBMS_SQL.COLUMN_VALUE_LONG(
cursorid                 IN    INTEGER,
position                 IN    INTEGER,
length                   IN    INTEGER,
off_set                  IN    INTEGER,
column_value            INOUT TEXT,
actual_length           INOUT INTEGER default 1024
)
RETURN RECORD;
```

Table 9-54 DBMS_SQL.COLUMN_VALUE_LONG interface parameters

Parameter Name	Description
cursorid	ID of the cursor to be executed
position	Position of a dynamically defined column in the query
length	Length of a return value
off_set	Start position of a return value
column_value	Return value
actual_length	Length of a return value

- DBMS_SQL.COLUMN_VALUE_RAW

  This function returns the value of the RAW type in a specified position of a cursor and accesses the data obtained by DBMS_SQL.FETCH_ROWS.

  The function prototype of **DBMS_SQL.COLUMN_VALUE_RAW** is:

```
DBMS_SQL.COLUMN_VALUE_RAW(
cursorid                IN    INTEGER,
position                IN    INTEGER,
column_value            INOUT BYTEA,
err_num                 INOUT NUMERIC default 0,
actual_length           INOUT INTEGER default 1024
)
RETURN RECORD;
```

Table 9-55 DBMS_SQL.COLUMN_VALUE_RAW interface parameters

Parameter Name	Description
cursorid	ID of the cursor to be executed
position	Position of a dynamically defined column in the query
column_value	Returned column value
err_num	Error No. It is an output parameter and the argument must be a variable. Currently, the output value is **–1** regardless of the argument.
actual_length	Length of a return value. The value longer than this length will be truncated.

- DBMS_SQL.COLUMN_VALUE_TEXT

This function returns the value of the TEXT type in a specified position of a cursor and accesses the data obtained by DBMS_SQL.FETCH_ROWS.

The function prototype of **DBMS_SQL.COLUMN_VALUE_TEXT** is:

```
DBMS_SQL.COLUMN_VALUE_TEXT(
cursorid                IN    INTEGER,
position                IN    INTEGER
)
RETURN TEXT;
```

Table 9-56 DBMS_SQL.COLUMN_VALUE_TEXT interface parameters

Parameter Name	Description
cursorid	ID of the cursor to be executed
position	Position of a dynamically defined column in the query

- DBMS_SQL.COLUMN_VALUE_UNKNOWN

  This function returns the value of an unknown type in a specified position of a cursor. This is an error handling interface when the type is not unknown.

  The function prototype of **DBMS_SQL.COLUMN_VALUE_UNKNOWN** is:

```
DBMS_SQL.COLUMN_VALUE_UNKNOWN(
cursorid                IN    INTEGER,
position                IN    INTEGER,
COLUMN_TYPE                IN    TEXT
)
RETURN TEXT;
```

Table 9-57 DBMS_SQL.COLUMN_VALUE_UNKNOWN interface parameters

Parameter Name	Description
cursorid	ID of the cursor to be executed
position	Position of a dynamically defined column in the query
column_type	Returned parameter type

- DBMS_SQL.IS_OPEN

This function returns the status of a cursor: **open**, **parse**, **execute**, or **define**. The value is **TRUE**. If the status is unknown, an error is reported. In other cases, the value is **FALSE**.

The function prototype of **DBMS_SQL.IS_OPEN** is:

```
DBMS_SQL.IS_OPEN(
cursorid                IN    INTEGER
)
RETURN BOOLEAN;
```

Table 9-58 DBMS_SQL.IS_OPEN interface parameters

Parameter Name	Description
cursorid	ID of the cursor to be queried

Examples

```
-- Perform operations on RAW data in a stored procedure.
create or replace procedure pro_dbms_sql_all_02(in_raw raw,v_in int,v_offset int)
as
cursorid int;
v_id int;
v_info bytea :=1;
query varchar(2000);
execute_ret int;
define_column_ret_raw bytea :='1';
define_column_ret int;
begin
drop table if exists pro_dbms_sql_all_tb1_02 ;
create table pro_dbms_sql_all_tb1_02(a int ,b blob);
insert into pro_dbms_sql_all_tb1_02 values(1,HEXTORAW('DEADBEEE'));
insert into pro_dbms_sql_all_tb1_02 values(2,in_raw);
query := 'select * from pro_dbms_sql_all_tb1_02 order by 1';
-- Open a cursor.
cursorid := dbms_sql.open_cursor();
-- Compile the cursor.
dbms_sql.parse(cursorid, query, 1);
-- Define a column.
define_column_ret:= dbms_sql.define_column(cursorid,1,v_id);
define_column_ret_raw:= dbms_sql.define_column_raw(cursorid,2,v_info,10);
-- Execute the cursor.
execute_ret := dbms_sql.execute(cursorid);
loop
exit when (dbms_sql.fetch_rows(cursorid) <= 0);
-- Obtain values.
dbms_sql.column_value(cursorid,1,v_id);
dbms_sql.column_value_raw(cursorid,2,v_info,v_in,v_offset);
-- Output the result.
dbms_output.put_line('id:'|| v_id || ' info:' || v_info);
end loop;
-- Close the cursor.
dbms_sql.close_cursor(cursorid);
end;
/
-- Invoke the stored procedure.
call pro_dbms_sql_all_02(HEXTORAW('DEADBEEF'),0,1);

-- Delete the stored procedure.
DROP PROCEDURE pro_dbms_sql_all_02;
```

9.11 GaussDB(DWS) Stored Procedure Debugging

Syntax

RAISE has the following five syntax formats:

Figure 9-34 raise_format::=

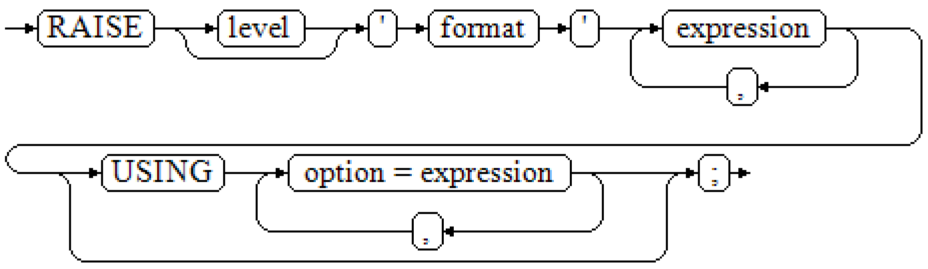


Figure 9-35 raise_condition::=

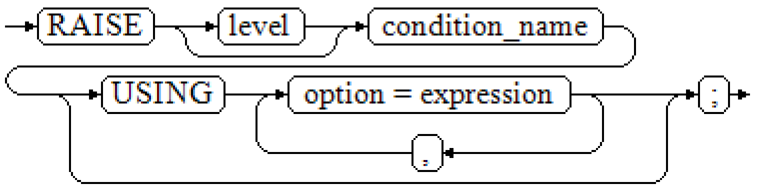


Figure 9-36 raise_sqlstate::=

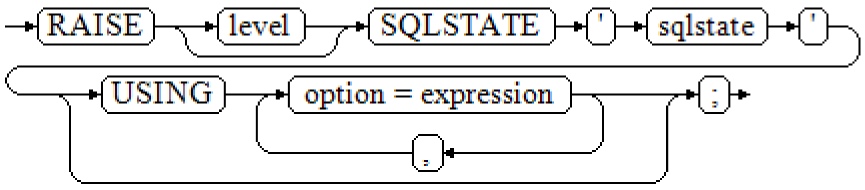


Figure 9-37 raise_option::=

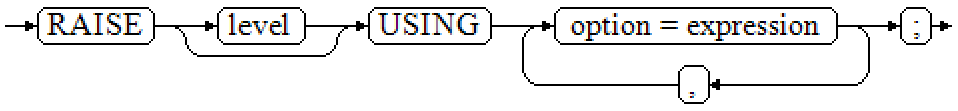
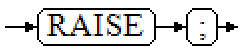


Figure 9-38 raise::=



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 character string. The format character string can be appended with an expression for

insertion to the message text. In a format character string, **%** is replaced by the parameter value attached to format and **%%** is used to print **%**. For example:

```
--v_job_id replaces % in the character 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 character 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**. Avoid using error codes that end in three zeros because they are category codes and can be captured by the entire category.

NOTE

The syntax described in **Figure 9-38** 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.

Examples

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);
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);

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);
ERROR:  division_by_zero
```

Alternatively:

```
RAISE unique_violation USING MESSAGE = 'Duplicate user ID: ' || user_id;
```

10 Using PostGIS Extension

10.1 PostGIS

NOTE

- The third-party software that the PostGIS Extension depends on needs to be installed separately. If you need to use PostGIS, submit a service ticket or contact technical support to submit an application.
- If the error message "ERROR: EXTENSION is not yet supported." is displayed, the PostGIS software package is not installed. Contact technical support.

GaussDB(DWS) provides PostGIS Extension (PostGIS-2.4.2). PostGIS Extension is a spatial database extender for PostgreSQL. It provides the following spatial information services: spatial objects, spatial indexes, spatial functions, and spatial operators. PostGIS Extension complies with the OpenGIS specifications.

In GaussDB(DWS), PostGIS Extension depends on the listed third-party open-source software.

- Geos 3.6.2
- Proj 4.9.2
- Json 0.12.1
- Libxml2 2.7.1
- Gdal 1.11.0

10.2 Using PostGIS

NOTE

- The third-party software that the PostGIS Extension depends on needs to be installed separately. If you need to use PostGIS, submit a service ticket or contact technical support to submit an application.
- If the error message "ERROR: EXTENSION is not yet supported." is displayed, the PostGIS software package is not installed. Contact technical support.
- The uuid-ossp extension has been preloaded in GaussDB(DWS). You can directly use the uuid function supported by GaussDB(DWS) without running the **CREATE EXTENSION uuid-ossp** command.

Creating PostGIS Extension

Run the **CREATE EXTENSION** command to create PostGIS Extension.

```
CREATE EXTENSION postgis;
```

Using PostGIS Extension

Use the following function to invoke a PostGIS Extension:

```
SELECT GisFunction (Param1, Param2,......);
```

GisFunction is the function, and **Param1** and **Param2** are function parameters. The following SQL statements are a simple illustration for PostGIS use. For details about related functions, see ***PostGIS 2.4.2 Manual***.

Example 1: Create a geometry table.

```
CREATE TABLE cities ( id integer, city_name varchar(50) );
SELECT AddGeometryColumn('cities', 'position', 4326, 'POINT', 2);
```

Example 2: Insert geometry data.

```
INSERT INTO cities (id, position, city_name) VALUES (1,ST_GeomFromText('POINT(-9.5 23)',4326),'CityA');
INSERT INTO cities (id, position, city_name) VALUES (2,ST_GeomFromText('POINT(-10.6 40.3)',4326),'CityB');
INSERT INTO cities (id, position, city_name) VALUES (3,ST_GeomFromText('POINT(20.8 30.3)',4326), 'CityC');
```

Example 3: Calculate the distance between any two cities among three cities.

```
SELECT p1.city_name,p2.city_name,ST_Distance(p1.position,p2.position) FROM cities AS p1, cities AS p2
WHERE p1.id > p2.id;
```

Deleting PostGIS Extension

Run the following command to delete PostGIS Extension from GaussDB(DWS):

```
DROP EXTENSION postgis [CASCADE];
```

NOTE

If PostGIS Extension is the dependee of other objects (for example, geometry tables), you need to add the **CASCADE** keyword to delete all these objects.

10.3 PostGIS Support and Constraints

Supported Data Types

In GaussDB(DWS), PostGIS Extension support the following data types:

- box2d
- box3d
- geometry_dump
- geometry
- geography
- raster

NOTE

If PostGIS is used by a user other than the creator of the PostGIS, set the following GUC parameters:

```
SET behavior_compat_options = 'bind_procedure_searchpath';
```

Supported Operators and Functions

Table 10-1 Operators and functions supported by PostGIS

Category	Function
Management functions	AddGeometryColumn, DropGeometryColumn, DropGeometryTable, PostGIS_Full_Version, PostGIS_GEOS_Version, PostGIS_Liblwgeom_Version, PostGIS_Lib_Build_Date, PostGIS_Lib_Version, PostGIS_PROJ_Version, PostGIS_Scripts_Build_Date, PostGIS_Scripts_Installed, PostGIS_Version, PostGIS_LibXML_Version, PostGIS_Scripts_Released, Populate_Geometry_Columns, UpdateGeometrySRID
Geometry constructors	ST_BdPolyFromText, ST_BdMPolyFromText, ST_Box2dFromGeoHash, ST_GeogFromText, ST_GeographyFromText, ST_GeogFromWKB, ST_GeomCollFromText, ST_GeomFromEWKB, ST_GeomFromEWKT, ST_GeometryFromText, ST_GeomFromGeoHash, ST_GeomFromGML, ST_GeomFromGeoJSON, ST_GeomFromKML, ST_GMLToSQL, ST_GeomFromText, ST_GeomFromWKB, ST_LineFromMultiPoint, ST_LineFromText, ST_LineFromWKB, ST_LinestringFromWKB, ST_MakeBox2D, ST_3DMakeBox, ST_MakeEnvelope, ST_MakePolygon, ST_MakePoint, ST_MakePointM, ST_MLineFromText, ST_MPointFromText, ST_MPolyFromText, ST_Point, ST_PointFromGeoHash, ST_PointFromText, ST_PointFromWKB, ST_Polygon, ST_PolygonFromText, ST_WKBToSQL, ST_WKTToSQL
Geometry accessors	GeometryType, ST_Boundary, ST_CoordDim, ST_Dimension, ST_EndPoint, ST_Envelope, ST_ExteriorRing, ST_GeometryN, ST_GeometryType, ST_InteriorRingN, ST_IsClosed, ST_IsCollection, ST_IsEmpty, ST_IsRing, ST_IsSimple, ST_IsValid, ST_IsValidReason, ST_IsValidDetail, ST_M, ST_NDims, ST_NPoints, ST_NRings, ST_NumGeometries, ST_NumInteriorRings, ST_NumInteriorRing, ST_NumPatches, ST_NumPoints, ST_PatchN, ST_PointN, ST_SRID, ST_StartPoint, ST_Summary, ST_X, ST_XMax, ST_XMin, ST_Y, ST_YMax, ST_YMin, ST_Z, ST_ZMax, ST_Zmflag, ST_ZMin

Category	Function
Geometry editors	ST_AddPoint, ST_Affine, ST_Force2D, ST_Force3D, ST_Force3DZ, ST_Force3DM, ST_Force4D, ST_ForceCollection, ST_ForceSFS, ST_ForceRHR, ST_LineMerge, ST_CollectionExtract, ST_CollectionHomogenize, ST_Multi, ST_RemovePoint, ST_Reverse, ST_Rotate, ST_RotateX, ST_RotateY, ST_RotateZ, ST_Scale, ST_Segmentize, ST_SetPoint, ST_SetSRID, ST_SnapToGrid, ST_Snap, ST_Transform, ST_Translate, ST_TransScale
Geometry outputs	ST_AsBinary, ST_AsEWKB, ST_AsEWKT, ST_AsGeoJSON, ST_AsGML, ST_AsHEXEWKB, ST_AsKML, ST_AsLatLonText, ST_AsSVG, ST_AsText, ST_AsX3D, ST_GeoHash
Operators	&&, &&&, &<, &<|, &>, <<, <<|, =, >>, @, |&>, |>>, ~, ~=, <->, <#>
Spatial relationships and measurements	ST_3DClosestPoint, ST_3DDistance, ST_3DDWithin, ST_3DDFullyWithin, ST_3DIntersects, ST_3DLongestLine, ST_3DMaxDistance, ST_3DShortestLine, ST_Area, ST_Azimuth, ST_Centroid, ST_ClosestPoint, ST_Contains, ST_ContainsProperly, ST_Covers, ST_CoveredBy, ST_Crosses, ST_LineCrossingDirection, ST_Disjoint, ST_Distance, ST_HausdorffDistance, ST_MaxDistance, ST_DistanceSphere, ST_DistanceSpheroid, ST_DFullyWithin, ST_DWithin, ST_Equals, ST_HasArc, ST_Intersects, ST_Length, ST_Length2D, ST_3DLength, ST_Length_Spheroid, ST_Length2D_Spheroid, ST_3DLength_Spheroid, ST_LongestLine, ST_OrderingEquals, ST_Overlaps, ST_Perimeter, ST_Perimeter2D, ST_3DPerimeter, ST_PointOnSurface, ST_Project, ST_Relate, ST_RelateMatch, ST_ShortestLine, ST_Touches, ST_Within
Geometry processing	ST_Buffer, ST_BuildArea, ST_Collect, ST_ConcaveHull, ST_ConvexHull, ST_CurveToLine, ST_DelaunayTriangles, ST_Difference, ST_Dump, ST_DumpPoints, ST_DumpRings, ST_FlipCoordinates, ST_Intersection, ST_LineToCurve, ST_MakeValid, ST_MemUnion, ST_MinimumBoundingCircle, ST_Polygonize, ST_Node, ST_OffsetCurve, ST_RemoveRepeatedPoints, ST_SharedPaths, ST_Shift_Longitude, ST_Simplify, ST_SimplifyPreserveTopology, ST_Split, ST_SymDifference, ST_Union, ST_UnaryUnion
Linear referencing	ST_LineInterpolatePoint, ST_LineLocatePoint, ST_LineSubstring, ST_LocateAlong, ST_LocateBetween, ST_LocateBetweenElevations, ST_InterpolatePoint, ST_AddMeasure
Miscellaneous functions	ST_Accum, Box2D, Box3D, ST_Expand, ST_Extent, ST_3Dextent, Find_SRID, ST_MemSize
Exceptional functions	PostGIS_AddBBox, PostGIS_DropBBox, PostGIS_HasBBox

Category	Function
Raster Management Functions	AddRasterConstraints, DropRasterConstraints, AddOverviewConstraints, DropOverviewConstraints, PostGIS_GDAL_Version, PostGIS_Raster_Lib_Build_Date, PostGIS_Raster_Lib_Version, and ST_GDALDrivers, and UpdateRasterSRID
Raster Constructors	ST_AddBand, ST_AsRaster, ST_Band, ST_MakeEmptyRaster, ST_Tile, and ST_FromGDALRaster
Raster Accessors	ST_GeoReference, ST_Height, ST_IsEmpty, ST_MetaData, ST_NumBands, ST_PixelHeight, ST_PixelWidth, ST_ScaleX, ST_ScaleY, ST_RasterToWorldCoord, ST_RasterToWorldCoordX, ST_RasterToWorldCoordY, ST_Rotation, ST_SkewX, ST_SkewY, ST_SRID, ST_Summary, ST_UpperLeftX, ST_UpperLeftY, ST_Width, ST_WorldToRasterCoord, ST_WorldToRasterCoordX, ST_WorldToRasterCoordY
Raster Band Accessors	ST_BandMetaData, ST_BandNoDataValue, ST_BandIsNoData, ST_BandPath, ST_BandPixelType, and ST_HasNoBand
Raster Pixel Accessors and Setters	ST_PixelAsPolygon, ST_PixelAsPolygons, ST_PixelAsPoint, ST_PixelAsPoints, ST_PixelAsCentroid, ST_PixelAsCentroids, ST_Value, ST_NearestValue, ST_Neighborhood, ST_SetValue, ST_SetValues, ST_DumpValues, and ST_PixelOfValue
Raster Editors	ST_SetGeoReference, ST_SetRotation, ST_SetScale, ST_SetSkew, ST_SetSRID, ST_SetUpperLeft, ST_Resample, ST_Rescale, ST_Reskew, and ST_SnapToGrid, ST_Resize, and ST_Transform
Raster Band Editors	ST_SetBandNoDataValue and ST_SetBandIsNoData
Raster Band Statistics and Analytics	ST_Count, ST_CountAgg, ST_Histogram, ST_Quantile, ST_SummaryStats, ST_SummaryStatsAgg, and ST_ValueCount
Raster Outputs	ST_AsBinary, ST_AsGDALRaster, ST_AsJPEG, ST_AsPNG, and ST_AsTIFF
Raster Processing	ST_Clip, ST_ColorMap, ST_Intersection, ST_MapAlgebra, ST_Reclass, and ST_Union ST_Distinct4ma, ST_InvDistWeight4ma, ST_Max4ma, ST_Mean4ma, ST_Min4ma, ST_MinDist4ma, ST_Range4ma, ST_StdDev4ma, and ST _Sum4ma, ST_Aspect, ST_HillShade, ST_Roughness, ST_Slope, ST_TPI, ST_TRI, Box3D, ST_ConvexHull, ST_DumpAsPolygons, and ST_ Envelope, ST_MinConvexHull, ST_Polygon, ST_Contains, ST_ContainsProperly, ST_Covers, ST_CoveredBy, ST_Disjoint, ST_Intersects, and ST_Overlaps, ST_Touches, ST_SameAlignment, ST_NotSameAlignmentRea-son, ST_Within, ST_DWithin, and ST_DFullyWithin

Category	Function
Raster Operators	&&, &<, &>, =, @, ~=, and ~

Spatial Indexes

In GaussDB(DWS), PostGIS Extension supports Generalized Search Tree (GIST) spatial indexes. This index type is inapplicable to partitioned tables. Different from B-tree indexes, GIS indexes are adaptable to all kinds of irregular data structures, which can effectively improve the retrieval efficiency for geometry and geographic data.

Run the following command to create a GiST index:

```
CREATE INDEX indexname ON tablename USING GIST ( geometryfield );
```

Extension Constraints

- Only row-store tables are supported.
- Only Oracle-compatible databases are supported.
- The topology object management module, Topology, is not supported.
- BRIN indexes are not supported.
- The **spatial_ref_sys** table can only be queried during scale-out.

10.4 OPEN SOURCE SOFTWARE NOTICE (For PostGIS)

Copyright (C) 2017 Sandro Santilli <strk@kbt.io>

Copyright (C) 2015 Sandro Santilli <strk@kbt.io>

Copyright (C) 2009 Paul Ramsey <pramsey@cleverelephant.ca>

Copyright (C) 2011 Sandro Santilli <strk@kbt.io>

Copyright 2010 Olivier Courtin <olivier.courtin@oslandia.com>

Copyright 2014 Nicklas Avén

Copyright 2011-2016 Regina Obe

Copyright (C) 2008 Paul Ramsey

Copyright (C) 2011-2015 Sandro Santilli <strk@kbt.io>

Copyright 2010-2012 Olivier Courtin <olivier.courtin@oslandia.com>

Copyright (C) 2015 Daniel Baston <dbaston@gmail.com>

Copyright (C) 2013 Nicklas Avén

Copyright (C) 2016 Sandro Santilli <strk@kbt.io>

Copyright 2017 Darafei Praliaskouski <me@komzpa.net>

Copyright (c) 2016, Paul Ramsey <pramsey@cleverelephant.ca>

Copyright (C) 2011-2012 Sandro Santilli <strk@kbt.io>

Copyright (C) 2011 Paul Ramsey <pramsey@cleverelephant.ca>

Copyright (C) 2007-2008 Mark Cave-Ayland

Copyright (C) 2001-2006 Refractions Research Inc.

Copyright 2015 Daniel Baston <dbaston@gmail.com>

Copyright 2009 David Skea <David.Skea@gov.bc.ca>

Copyright (C) 2012-2015 Paul Ramsey <pramsey@cleverelephant.ca>

Copyright (C) 2012-2015 Sandro Santilli <strk@kbt.io>

Copyright 2001-2006 Refractions Research Inc.

Copyright (C) 2004 Refractions Research Inc.

Copyright 2011-2014 Sandro Santilli <strk@kbt.io>

Copyright 2009-2010 Sandro Santilli <strk@kbt.io>

Copyright 2015-2016 Daniel Baston <dbaston@gmail.com>

Copyright 2011-2015 Sandro Santilli <strk@kbt.io>

Copyright 2007-2008 Mark Cave-Ayland

Copyright 2012-2013 Oslandia <infos@oslandia.com>

Copyright (C) 2015-2017 Sandro Santilli <strk@kbt.io>

Copyright (C) 2001-2003 Refractions Research Inc.

Copyright (c) 1999, Frank Warmerdam

Copyright 2009 Mark Cave-Ayland <mark.cave-ayland@siriusit.co.uk>

Copyright (c) 2007, Frank Warmerdam

Copyright 2008 OpenGeo.org

Copyright (C) 2008 OpenGeo.org

Copyright (C) 2009 Mark Cave-Ayland <mark.cave-ayland@siriusit.co.uk>

Copyright 2010 LISAsoft

Copyright (C) 2010 Mark Cave-Ayland <mark.cave-ayland@siriusit.co.uk>

Copyright (c) 1999, 2001, Frank Warmerdam

Copyright (C) 2016-2017 Bj?rn Harrtell <bjorn@wololo.org>

Copyright (C) 2017 Danny G?tte <danny.goette@fem.tu-ilmenau.de>

Copyright 2009-2011 Paul Ramsey <pramsey@cleverelephant.ca>

^copyright^

Copyright 2012 (C) Paul Ramsey <pramsey@cleverelephant.ca>

Copyright (C) 2006 Refractions Research Inc.

Copyright 2009 Paul Ramsey <pramsey@opengeo.org>

Copyright 2001-2009 Refractions Research Inc.

Copyright (C) 2010 Olivier Courtin <olivier.courtin@oslandia.com>

By Nathan Wagner, copyright disclaimed,

this entire file is in the public domain

Copyright 2009-2011 Olivier Courtin <olivier.courtin@oslandia.com>

Copyright (C) 2001-2005 Refractions Research Inc.

Copyright 2001-2011 Refractions Research Inc.

Copyright 2009-2014 Sandro Santilli <strk@kbt.io>

Copyright (C) 2008 Paul Ramsey <pramsey@cleverelephant.ca>

Copyright (C) 2007 Refractions Research Inc.

Copyright (C) 2010 Sandro Santilli <strk@kbt.io>

Copyright 2012 J Smith <dark.panda@gmail.com>

Copyright 2009 - 2010 Oslandia

Copyright 2009 Oslandia

Copyright 2001-2005 Refractions Research Inc.

Copyright 2016 Paul Ramsey <pramsey@cleverelephant.ca>

Copyright 2016 Daniel Baston <dbaston@gmail.com>

software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another

language. (Hereinafter, translation is included without limitation in the term "modification".) Each license is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and

all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you

Software:Geos

Copyright notice:

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2012 Excensus LLC.

Copyright (C) 1996-2015 Free Software Foundation, Inc.

Copyright (c) 1995 Olivier Devillers <Olivier.Devillers@sophia.inria.fr>

Copyright (C) 2007-2010 Safe Software Inc.

Copyright (C) 2010 Safe Software Inc.

Copyright (C) 2006 Refractions Research

Copyright 2004 Sean Gillies, sgillies@frii.com

Copyright (C) 2011 Mateusz Loskot <mateusz@loskot.net>

Copyright (C) 2015 Nyall Dawson <nyall dot dawson at gmail dot com>

Original code (2.0 and earlier)copyright (c) 2000-2006 Lee Thomason (www.grinninglizard.com)

Original code (2.0 and earlier)copyright (c) 2000-2002 Lee Thomason (www.grinninglizard.com)

License: LGPL V2.1

GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Copyright (C) 2005-2011 Refractions Research Inc.

Copyright (C) 2009 Ragi Y. Burhum <ragi@burhum.com>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2005 2006 Refractions Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006-2011 Refractions Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net

Copyright (C) 2009-2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2016 Daniel Baston

Copyright (C) 2008 Sean Gillies

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006 Refractions Research Inc.

Copyright (C) 2005 2006 Refractions Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006-2011 Refractions Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net

Copyright (C) 2009-2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2016 Daniel Baston

Copyright (C) 2008 Sean Gillies

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006 Refractions Research Inc.

Copyright (C) 2012 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2008-2010 Safe Software Inc.

Copyright (C) 2006-2007 Refractions Research Inc.

Copyright (C) 2005-2007 Refractions Research Inc.

Copyright (C) 2007 Refractions Research Inc.

Copyright (C) 2014 Mika Heiskanen <mika.heiskanen@fmi.fi>

Copyright (C) 2009-2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 Mateusz Loskot

Copyright (C) 2005-2009 Refractions Research Inc.

Copyright (C) 2001-2009 Vivid Solutions Inc.

Copyright (C) 2012 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006 Wu Yongwei

Copyright (C) 2012 Excensus LLC.

Copyright (C) 1996-2015 Free Software Foundation, Inc.

Copyright (c) 1995 Olivier Devillers <Olivier.Devillers@sophia.inria.fr>

Copyright (C) 2007-2010 Safe Software Inc.

Copyright (C) 2010 Safe Software Inc.

Copyright (C) 2006 Refractions Research

Copyright 2004 Sean Gillies, sgillies@frii.com

Copyright (C) 2011 Mateusz Loskot <mateusz@loskot.net>

Copyright (C) 2015 Nyall Dawson <nyall dot dawson at gmail dot com>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 Mateusz Loskot

Copyright (C) 2005-2009 Refractions Research Inc.

Copyright (C) 2001-2009 Vivid Solutions Inc.

Copyright (C) 2012 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006 Wu Yongwei

Copyright (C) 2012 Excensus LLC.

Copyright (C) 1996-2015 Free Software Foundation, Inc.

Copyright (c) 1995 Olivier Devillers <Olivier.Devillers@sophia.inria.fr>

Copyright (C) 2007-2010 Safe Software Inc.

Copyright (C) 2010 Safe Software Inc.

Copyright (C) 2006 Refractions Research

Copyright 2004 Sean Gillies, sgillies@frii.com

Copyright (C) 2011 Mateusz Loskot <mateusz@loskot.net>

Copyright (C) 2015 Nyall Dawson <nyall dot dawson at gmail dot com>

Original code (2.0 and earlier)copyright (c) 2000-2006 Lee Thomason (www.grinninglizard.com)

Original code (2.0 and earlier)copyright (c) 2000-2002 Lee Thomason (www.grinninglizard.com)

License: LGPL V2.1

GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Copyright (C) 2005-2011 Refractions Research Inc.

Copyright (C) 2009 Ragi Y. Burhum <ragi@burhum.com>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2005 2006 Refractions Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006-2011 Refractions Research Inc.

Copyright (C) 2005-2011 Refractions Research Inc.

Copyright (C) 2009 Ragi Y. Burhum <ragi@burhum.com>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2005 2006 Refractions Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006-2011 Refractions Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net

Copyright (C) 2009-2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2016 Daniel Baston

Copyright (C) 2008 Sean Gillies

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006 Refractions Research Inc.

Copyright (C) 2012 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2008-2010 Safe Software Inc.

Copyright (C) 2006-2007 Refractions Research Inc.

Copyright (C) 2005-2007 Refractions Research Inc.

Copyright (C) 2007 Refractions Research Inc.

Copyright (C) 2014 Mika Heiskanen <mika.heiskanen@fmi.fi>

Copyright (C) 2009-2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 Mateusz Loskot

Copyright (C) 2005-2009 Refractions Research Inc.

Copyright (C) 2001-2009 Vivid Solutions Inc.

Copyright (C) 2012 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006 Wu Yongwei

Copyright (C) 2012 Excensus LLC.

Copyright (C) 1996-2015 Free Software Foundation, Inc.

Copyright (c) 1995 Olivier Devillers <Olivier.Devillers@sophia.inria.fr>

Copyright (C) 2007-2010 Safe Software Inc.

Copyright (C) 2010 Safe Software Inc.

Copyright (C) 2007 Refractions Research Inc.

Copyright (C) 2014 Mika Heiskanen <mika.heiskanen@fmi.fi>

Copyright (C) 2009-2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 Mateusz Loskot

Copyright (C) 2005-2009 Refractions Research Inc.

Copyright (C) 2001-2009 Vivid Solutions Inc.

Copyright (C) 2012 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006 Wu Yongwei

Copyright (C) 2012 Excensus LLC.

Copyright (C) 1996-2015 Free Software Foundation, Inc.

Copyright (c) 1995 Olivier Devillers <Olivier.Devillers@sophia.inria.fr>

Copyright (C) 2007-2010 Safe Software Inc.

Copyright (C) 2010 Safe Software Inc.

Copyright (C) 2006 Refractions Research

Copyright 2004 Sean Gillies, sgillies@frii.com

Copyright (C) 2011 Mateusz Loskot <mateusz@loskot.net>

Copyright (C) 2015 Nyall Dawson <nyall dot dawson at gmail dot com>

Original code (2.0 and earlier)copyright (c) 2000-2006 Lee Thomason (www.grinninglizard.com)

Original code (2.0 and earlier)copyright (c) 2000-2002 Lee Thomason (www.grinninglizard.com)

License: LGPL V2.1

GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Copyright (C) 2005-2011 Refractions Research Inc.

Copyright (C) 2009 Ragi Y. Burhum <ragi@burhum.com>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2015 Nyall Dawson <nyall dot dawson at gmail dot com>

Original code (2.0 and earlier)copyright (c) 2000-2006 Lee Thomason (www.grinninglizard.com)

Original code (2.0 and earlier)copyright (c) 2000-2002 Lee Thomason (www.grinninglizard.com)

License: LGPL V2.1

GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

[This is the first released version of the Lesser GPL. It also counts as the successor of the GNU Library Public License, version 2, hence the version number 2.1.]

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public

Licenses are intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software packages--typically libraries--of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and

is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the "Lesser" General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The

former contains code derived from the library, whereas the latter must be combined with the library in order to run.

GNU LESSER GENERAL PUBLIC LICENSE

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under

copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an

appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the

Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1

above, provided that you also meet all of these conditions:

a) The modified work must itself be a software library.

b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of

its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in

themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or

collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany

it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to

distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a

work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License.

Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object

file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6,

whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work

under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

b) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system,

rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.

c) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

d) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

e) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception,

the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on

which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot

use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this

License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein.

You are not responsible for enforcing compliance by third parties with this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the

integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot

impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time.

Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is

copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status

of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING

RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the library's name and a brief idea of what it does.>

Copyright (C) <year> <name of author>

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU

Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the library, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the library `Frob' (a library for tweaking knobs) written by James Random Hacker.

<signature of Ty Coon>, 1 April 1990

Ty Coon, President of Vice

That's all there is to it!

Software: JSON-C

Copyright notice:

Copyright (c) 2004, 2005 Metaparadigm Pte. Ltd.

Copyright (c) 2009-2012 Eric Haszlakiewicz

Copyright (c) 2004, 2005 Metaparadigm Pte Ltd

Copyright (c) 2009 Hewlett-Packard Development Company, L.P.

Copyright (c) 2011, 2012 Martin Lambers <marlam@marlam.de>

Copyright (c) 2006, Andrey Kiselev

Copyright (c) 2008-2012, Even Rouault <even dot rouault at mines-paris dot org>

Copyright (c) 2001, Frank Warmerdam

Copyright (c) 2001, Frank Warmerdam <warmerdam@pobox.com>

Copyright (c) 2008 Gerald I. Evenden

"

License: MIT License

Please see above

Software: libxml2

Copyright notice:

"See Copyright for the status of this software.

Copyright (C) 1998-2003 Daniel Veillard. All Rights Reserved.

Copyright (C) 2003 Daniel Veillard.

copy: see Copyright for the status of this software.

copy: see Copyright for the status of this software

copy: see Copyright for the status of this software.

Copyright (C) 2000 Bjorn Reese and Daniel Veillard.

Copy: See Copyright for the status of this software.

See COPYRIGHT for the status of this software

Copyright (C) 2000 Gary Pennington and Daniel Veillard.

Copyright (C) 1996, 1997, 1998, 1999, 2000, 2001, 2003, 2004, 2005, 2006,

2007 Free Software Foundation, Inc.

Copyright (C) 1998 Bjorn Reese and Daniel Stenberg.

Copyright (C) 2001 Bjorn Reese <breese@users.sourceforge.net>

Copyright (C) 2000 Bjorn Reese and Daniel Stenberg.

Copyright (C) 2001 Bjorn Reese and Daniel Stenberg.

See Copyright for the status of this software

"

License: MIT License

Please see above

11 Using JDBC or ODBC for GaussDB(DWS) Secondary Development

11.1 Prerequisites

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

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

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

Downloading Drivers

For details, see **Downloading the JDBC or ODBC Driver**.

11.2 JDBC-Based Development

11.2.1 JDBC Development Process

Java Database Connectivity (JDBC) is a Java API for executing SQL statements. It provides a unified access interface for multiple relational databases, enabling applications to work with data based on it. GaussDB(DWS) supports JDBC 4.0 and requires JDK 1.6 or later for code compiling. It does not support JDBC-ODBC Bridge. The following figure shows the JDBC application development process.

Figure 11-1 JDBC-based application development process

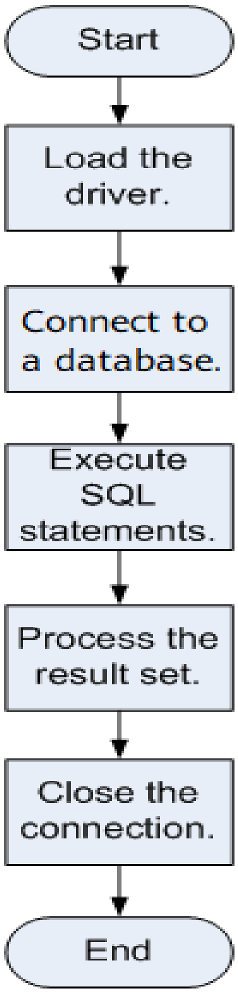


Table 11-1 JDBC development process

Step	Description
Load the driver.	Download the JDBC driver and edit and load it in the program.
Connect to a database.	Connect to the database through the JDBC driver.
Execute SQL statements.	Applications operate database data by executing SQL statements.
Process the result set.	Different types of result sets have different application scenarios. Applications need to select the appropriate result set type as needed.
Close the connection.	Make sure to close the database connection after completing the required data operations.

11.2.2 JDBC Package and Driver Class

JDBC Package

Download the **dws_8.1.x_jdbc_driver.zip** package from the console.

For details, see **Downloading the JDBC or ODBC Driver**.

After the decompression, you will obtain the following JDBC packages in .jar format:

- **gsjdbc4.jar**: Driver package compatible with PostgreSQL. The class name and class structure in the driver are the same as those in the PostgreSQL driver. All the applications running on PostgreSQL can be smoothly transferred to the current system.
- **gsjdbc200.jar**: This driver package is used when both PostgreSQL and GaussDB(DWS) are accessed in a JVM process. The main class name is **com.huawei.gauss200.jdbc.Driver** and the prefix of the URL for database connection is **jdbc:gaussdb**. Other information of this driver package is the same as that of **gsjdbc4.jar**.

Driver Class

Before creating a database connection, you need to load the database driver class **org.postgresql.Driver** (decompressed from **gsjdbc4.jar**) or **com.huawei.gauss200.jdbc.Driver** (decompressed from **gsjdbc200.jar**).

NOTE

GaussDB(DWS) is compatible with PostgreSQL in the use of JDBC. Therefore, when two JDBC drivers are used in the same process, class names may conflict.

11.2.3 Loading a Driver

Load the database driver before creating a database connection.

You can load the driver in the following ways:

- Implicitly loading the driver before creating a connection in the code: **Class.forName ("org.postgresql.Driver")**
- Transferring a parameter during the JVM startup: **java -Djdbc.drivers=org.postgresql.Driver jdbctest**

  NOTE

  - **jdbctest** is the name of a test application.
  - If **gsjdbc200.jar** is used, change the driver class name to **"com.huawei.gauss200.jdbc.Driver"**.

11.2.4 Connecting to a Database

After a database is connected, you can run SQL statements the database to perform operations on data.

NOTE

If you use an open-source Java Database Connectivity (JDBC) driver, ensure that the database parameter **password_encryption_type** is set to **1**. If the value is not 1, the connection may fail. A typical error message is "none of the server's SASL authentication mechanisms are supported." To avoid such problems, perform the following operations:

1. Set **password_encryption_type** to **1**. For details, see **Modifying Database Parameters**.
2. Create a new database user for connection or reset the password of the existing database user.
   - If you use an administrator account, reset the password. For details, see **Resetting a Password**.
   - If you are a common user, use another client tool (such as Data Studio) to connect to the database and run the **ALTER USER** statement to change your password.
3. Connect to the database.

Here are the reasons why you need to perform these operations:

- MD5 algorithms may by vulnerable to collision attacks and cannot be used for password verification. Currently, GaussDB(DWS) uses the default security design. By default, MD5 password verification is disabled, but MD5 is required by the open-source libpq communication protocol of PostgreSQL. For connectivity purposes, you need to adjust the cryptographic algorithm parameter **password_encryption_type** and enable the MD5 algorithm.
- The database stores the hash digest of passwords instead of password text. During password verification, the system compares the hash digest with the password digest sent from the client (salt operations are involved). If you change your cryptographic algorithm policy, the database cannot generate a new MD5 hash digest for your existing password. For connectivity purposes, you must manually change your password or create a new user. The new password will be encrypted using the hash algorithm and stored for authentication in the next connection.

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 11-2 Database connection parameters

Parameters	Description
url	**gsjdbc4.jar** database connection descriptor. The descriptor format can be: • jdbc:postgresql:database • jdbc:postgresql://host/database • jdbc:postgresql://host:port/database • jdbc:postgresql://host:port[,host:port][...]/database **NOTE** If **gsjdbc200.jar** is used, replace **jdbc:postgresql** with **jdbc:gaussdb**. • **database**: indicates the name of the database to be connected. • **host** indicates the name or IP address of the database server. If an ELB is bound to the cluster, set **host** to the IP address of the ELB. • **port**: indicates the port number of a database server. By default, the database on port **8000** of the local host is connected. • Multiple IP addresses and ports can be configured. JDBC balances load by random access and failover, and will automatically ignore unreachable IP addresses. IP addresses are separated using commas. Example: **jdbc:postgresql://10.10.0.13:8000,10.10.0.14:8000/database** • If JDBC is used to connect to a cluster, only JDBC connection parameters can be configured in a cluster address. Variables cannot be added.

Parameters	Description
info	Database connection properties. Common properties include: ● **user**: string type. It indicates the database user establishing a connection. ● **password**: string type. It indicates the password of a database user. ● **ssl**: Boolean type. It indicates whether the Secure Socket Layer (SSL) is used. ● **loggerLevel**: string type. It indicates the amount of information that the driver logs and prints to the LogStream or LogWriter specified in the DriverManager. Currently, **OFF**, **DEBUG**, and **TRACE** are supported. **DEBUG** indicates that only logs of the **DEBUG** or higher level are printed, generating a few log information. **TRACE** indicates that logs of the **DEBUG** and **TRACE** levels are printed, generating detailed log information. The default value is **OFF**, indicating that no information will be logged. ● **prepareThreshold**: integer type. It indicates the number of **PreparedStatement** executions required before SQL statements are switched over to servers as prepared statements. The default value is **5**. ● **batchMode**: boolean type. It indicates whether to connect the database in batch mode. ● **fetchsize**: integer type. It indicates the default fetchsize for statements in the created connection. ● **ApplicationName**: string type. It indicates an application name. The default value is **PostgreSQL JDBC Driver**. ● **allowReadOnly**: boolean type. It indicates whether to enable the read-only mode for connection. The default value is **false**. If the value is not changed to **true**, the execution of **connection.setReadOnly** does not take effect. ● **blobMode**: string type. It is used to set the setBinaryStream method to assign values to different data types. The value **on** indicates that values are assigned to the BLOB data type and **off** indicates that values are assigned to the bytea data type. The default value is **on**. ● **connectionExtraInfo**: boolean type. It indicates whether the JDBC driver reports the driver deployment path and process owner to the database. **NOTE** The value can be **true** or **false**. The default value is **true**. If **connectionExtraInfo** is set to **true**, the JDBC 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** or **PGXC_STAT_ACTIVITY**.
user	Indicates a database user.

Parameters	Description
password	Indicates the password of a database user.

Closing the Connection

Make sure to close the database connection after completing the required data operations.

To close the database connection, you can directly invoke the **close** method, for example, **conn.close()**.

Examples

```
//gsjdbc4.jar is used as an example. If gsjdbc200.jar is used, replace the driver class name org.postgresql
with com.huawei.gauss200.jdbc and replace the URL prefix jdbc:postgresql with jdbc:gaussdb.
//The following code encapsulates database connection operations into an interface. The database can then
be connected using an authorized username and password.

public static Connection GetConnection(String username, String passwd) {
        //Set the driver class.
        String driver = "org.postgresql.Driver";
        //Database connection descriptor.
        String sourceURL = "jdbc:postgresql://10.10.0.13:8000/postgres?currentSchema=test";
        Connection conn = null;

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

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

        return conn;
    }
```

11.2.5 Executing SQL Statements

Executing an Ordinary SQL Statement

The application performs data (parameter statements do not need to be transferred) in the database by running SQL statements, and you need to perform the following steps:

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

```
Statement stmt = con.createStatement();
```

Step 2 Execute the SQL statement by triggering 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.

Step 3 Close the statement object.

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

----End

Executing a Prepared SQL Statement

Pre-compiled statements were once complied and optimized and can have additional parameters for different usage. For the statements have been pre-compiled, the execution efficiency is greatly improved. If you want to execute a statement for several times, use a precompiled statement. Perform the following procedure:

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 triggering the setShort method in PreparedStatement.

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

Step 3 Execute the precompiled SQL statement by triggering the executeUpdate method in PreparedStatement.

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

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

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

----End

Calling a Stored Procedure

Perform the following steps to call existing stored procedures through the JDBC interface in GaussDB(DWS):

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

```
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 with an output parameter by calling the registerOutParameter method in CallableStatement.

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

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.
```

For example:

```
//The following stored procedure has been created with the OUT parameter:
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. Close these actions 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 a 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(DWS) JDBC driver does not support this method.
- GaussDB(DWS) does not support functions containing OUT parameters, or default values of stored procedures and function parameters.

----End

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 executed separately.

Batch Processing

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

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

```
PreparedStatement pstmt = con.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 Execute batch processing by calling the executeBatch method in PreparedStatement.

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

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

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

NOTE

Do not terminate a batch processing action when it is ongoing; otherwise, the database performance will deteriorate. Therefore, disable the automatic submission function during batch processing, and manually submit every several lines. The statement for disabling automatic submission is **conn.setAutoCommit(false)**.

----End

11.2.6 Processing Data in a Result Set

Setting a Result Set Type

Different types of result sets are applicable to different application scenarios. Applications select proper types of result sets based on requirements. Before executing an SQL statement, you must create a statement object. Some methods of creating statement objects can set the type of a result set. **Table 11-3** 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 11-3 Result set types

Parameter	Description
resultSetType	Indicates the 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.
resultSetConcurren-cy	Indicates the concurrency type of a result set. There are two types of concurrency. ● **ResultSet.CONCUR_READ_ONLY**: The 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 11-4** lists these methods.

Table 11-4 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.

Method	Description
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 cursor forward or backward a specified number of rows.

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. JDBC driver provides a method to obtain the cursor position in a result set. **Table 11-5** lists the method.

Table 11-5 Method for obtaining the 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 11-6** lists the common methods for obtaining data. If you want to know more about other methods, see JDK official documents.

Table 11-6 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 int.
int getInt(String columnLabel)	Retrieves the value of the column designated by a column label in the current row as an int.
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.
Date getDate(String columnLabel)	Retrieves the value of the column designated by a column name in the current row as a Date.

11.2.7 Common JDBC Development Examples

Example 1

Before completing the following example, you need to create a stored procedure.

```
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;
/
```

This example illustrates how to develop applications based on the GaussDB(DWS) JDBC interface.

```
//DBtest.java
//gsjdbc4.jar is used as an example. If gsjdbc200.jar is used, replace the driver class name org.postgresql with com.huawei.gauss200.jdbc and replace the URL prefix jdbc:postgresql with jdbc:gaussdb.
// 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;

public class DBTest {

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

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

    return conn;
  };

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

      //Run an ordinary 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 the preprocessing 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();
      }
      //Run batch processing.
      pst.executeBatch();
      pst.close();
    } catch (SQLException e) {
      if (pst != null) {
        try {
          pst.close();
        } catch (SQLException e1) {
        e1.printStackTrace();
        }
      }
      e.printStackTrace();
    }
  }

  //Run the precompilation 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 {

      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 as an integer.
      cstmt.execute();
      int out = cstmt.getInt(4);  //Obtain the out parameter value.
      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) {
    //Establish a connection to the database.
    Connection conn = GetConnection("tester", "password");
    //Create a table.
    CreateTable(conn);

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

  //Run the precompilation 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, you need to disable the autocommit function.

```
// Disable the autocommit function.
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.");
}
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.");
}
rs.close();

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

Retrying SQL Queries for Applications

If the primary DN is faulty and cannot be restored within 40 seconds, its standby is automatically promoted to primary to ensure that the cluster runs properly. Jobs running during the switchover will fail and those started after the switchover will not be affected. To protect upper-layer services from being affected by the failover, refer to the following example to construct a SQL retry mechanism at the service layer.

gsjdbc4.jar is used as an example. If **gsjdbc200.jar** is used, replace the driver class name **org.postgresql** with **com.huawei.gauss200.jdbc** and replace the URL prefix **jdbc:postgresql** with **jdbc:gaussdb**.

```
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 {
  //Establish a connection to the database.
   public static Connection GetConnection(String username, String passwd) {
     String driver = "org.postgresql.Driver";
     String sourceURL = "jdbc:postgresql://10.131.72.136:8000/gaussdb";
     Connection conn = null;
     try {
      //Load the database driver.
       Class.forName(driver).newInstance();
     } catch (Exception e) {
       e.printStackTrace();
       return null;
     }

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

     return conn;
}
```

Run an ordinary SQL statement. Create the **jdbc_test1** table.

```
public static void CreateTable(Connection conn) {
     Statement stmt = null;
     try {
       stmt = conn.createStatement();

       // add ctrl+c handler
       Runtime.getRuntime().addShutdownHook(new ExitHandler(stmt));
```

```
      //Run an ordinary 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 the preprocessing 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();
       }
      //Run batch processing.
       pst.executeBatch();
       pst.close();
     } catch (SQLException e) {
       if (pst != null) {
         try {
           pst.close();
         } catch (SQLException e1) {
         e1.printStackTrace();
         }
       }
       e.printStackTrace();
     }
   }
```

Run the precompilation 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;
   }
```

Run a query statement and retry upon a failure. The number of retry times can be configured.

```
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 {
 //Establish a connection to the database.
    Connection conn = GetConnection("testuser", "test@123");

   //Create a table.
    CreateTable(conn);

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

 //Run the precompilation statement to update data.
    ExecPreparedSQL(conn);

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

  }

}
```

Importing and Exporting Data Through Local Files

When the JAVA language is used for secondary development based on GaussDB(DWS), 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(DWS) JDBC driver before running it.

gsjdbc4.jar is used as an example. If **gsjdbc200.jar** is used, replace the driver class name **org.postgresql** with **com.huawei.gauss200.jdbc** and replace the URL prefix **jdbc:postgresql** with **jdbc:gaussdb**.

```
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/gaussdb"); //URL of the database
      String username = new String("jack");             //Username
      String password = new String("********");        //Password
      String tablename = new String("migration_table"); //Define table information.
      String tablename1 = new String("migration_table_1"); //Define 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);
          }
```

Import and export data.

```
      //Export the query result of migration_table to the local file d:/data.txt.
      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", migration_table_1);
  } catch (SQLException e) {
 // TODO Auto-generated catch block
        e.printStackTrace();
} catch (IOException e) {
 // TODO Auto-generated catch block
 e.printStackTrace();
}
```

```
      //Export the data from the migration_table_1 table to the d:/data1.txt file.
      try {
      copyToFile(conn, "d:/data1.txt", migration_table_1);
   } 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();
               }
           }
       }
  }
}
```

Migrating Data from MySQL to GaussDB(DWS)

The following example shows how to use CopyManager to migrate data from MySQL to GaussDB(DWS).

gsjdbc4.jar is used as an example. If **gsjdbc200.jar** is used, replace the driver class name **org.postgresql** with **com.huawei.gauss200.jdbc** and replace the URL prefix **jdbc:postgresql** with **jdbc: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://10.180.155.74:8000/gaussdb"); //URL of the database
        String user = new String("jack");             //GaussDB(DWS) username
        String pass = new String("********");             //GaussDB(DWS) password
        String tablename = new String("migration_table"); //Define table information.
        String delimiter = new String("|");              //Define a delimiter.
        String encoding = new String("UTF8");            //Define a character set.
        String driver = "org.postgresql.Driver";
        StringBuffer buffer = new StringBuffer();       //Define the 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
newline character to form 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 table information.
                String sql = "Copy " + tablename + " from STDIN DELIMITER " + "'" + delimiter + "'" + "
ENCODING " + "'" + encoding + "'";

                //Submit 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 from the source database.

```
    private static ResultSet getDataSet() {
        ResultSet rs = null;
        try {
            Class.forName("com.mysql.jdbc.Driver").newInstance();
            Connection conn = DriverManager.getConnection("jdbc:mysql://10.119.179.227:3306/jack?
useSSL=false&allowPublicKeyRetrieval=true", "jack", "********");
            Statement stmt = conn.createStatement();
            rs = stmt.executeQuery("select * from migration_table");
        } catch (SQLException e) {
            e.printStackTrace();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return rs;
    }
}
```

11.2.8 Processing RoaringBitmap Result Sets and Importing It to GaussDB (DWS)

GaussDB(DWS) 8.1.3 and later versions support the RoaringBitmap function. When using the Java language to perform secondary development based on GaussDB(DWS), you can use the CopyManager interface to import a small amount of RoaringBitmap data to GaussDB(DWS).

NOTE

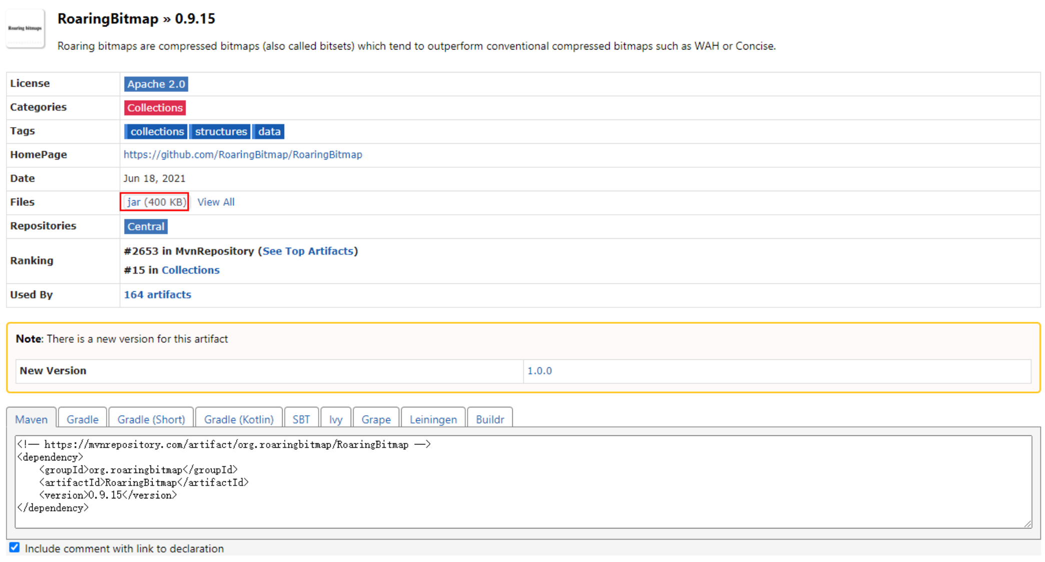
To import a large amount of RoaringBitmap data, computing power of the application side needs to be increased. Otherwise, the import performance will be affected.

Processing RoaringBitmap Data

Step 1 Visit **Maven** to download the open-source RoaringBitmap JAR package. Version 0.9.15 is recommended.

The dependency items of the POM file are configured as follows:

```
<dependencies>
 <dependency>
 <groupId>org.roaringbitmap</groupId>
 <artifactId>RoaringBitmap</artifactId>
 <version>0.9.15</version>
 </dependency>
 </dependencies>
```



RoaringBitmap » 0.9.15

Roaring bitmaps are compressed bitmaps (also called bitsets) which tend to outperform conventional compressed bitmaps such as WAH or Concise.

License	Apache 2.0
Categories	Collections
Tags	collections structures data
HomePage	https://github.com/RoaringBitmap/RoaringBitmap
Date	Jun 18, 2021
Files	jar (400 KB) View All
Repositories	Central
Ranking	#2653 in MvnRepository (See Top Artifacts) #15 in Collections
Used By	164 artifacts

Note: There is a new version for this artifact

New Version	1.0.0

```
<!-- https://mvnrepository.com/artifact/org.roaringbitmap/RoaringBitmap -->
<dependency>
    <groupId>org.roaringbitmap</groupId>
    <artifactId>RoaringBitmap</artifactId>
    <version>0.9.15</version>
</dependency>
```

☑ Include comment with link to declaration

Step 2 Invoke the JAR package to convert data to the RoaringBitmap type.

The general process is to declare a Roaring bitmap, call the add() method to convert data of the int type into the Roaringbitmap type, and then serialize the converted data. The sample code is as follows:

```
RoaringBitmap rr2 = new RoaringBitmap ();
for (int i = 1; i < 10000000;  i++)  {
      rr2.add(i);
}
ByteArrayOutputStream a = new ByteArrayOutputStream();
DataOutputStream b = new DataOutputStream(a);
rr2.serialize(b);
```

----End

Data Import

Invoke CopyManager to import data to the database. In this way, a small amount of RoaringBitmap data can be imported to the database without having to be stored locally.

```
//gsjdbc4.jar is used as an example. If gsjdbc200.jar is used, replace the driver class name org.postgresql with com.huawei.gauss200.jdbc and replace the URL prefix jdbc:postgresql with jdbc:gaussdb.

package rb_demo;

import org.postgresql.copy.CopyManager;
import org.postgresql.core.BaseConnection;
import org.roaringbitmap.RoaringBitmap;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.StringReader;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class rb_demo {

    private static String hexStr =  "0123456789ABCDEF";

    public static String bytesToHex(byte[] bytes) {
        StringBuffer sb = new StringBuffer();
        for (int i = 0; i < bytes.length; i++) {
            String hex = Integer.toHexString(bytes[i] & 0xFF);
            if (hex.length() < 2) {
                sb.append(0);
            }
            sb.append(hex);
        }
        return sb.toString();
    }

    public static Connection GetConnection(String username, String passwd) {
        String driver = "org.postgresql.Driver";
String sourceURL = "jdbc:postgresql://10.185.180.161: 8000/gaussdb"; //Database URL
        Connection conn = null;
        try {
      //Load the database driver.
            Class.forName(driver).newInstance();
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }

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

        return conn;
    }

    public static void main(String[] args) throws IOException {

        RoaringBitmap rr2 = new RoaringBitmap();

        for (int i = 1; i < 10000000; i++) {
            rr2.add(i);
        }

        ByteArrayOutputStream a = new ByteArrayOutputStream();

        DataOutputStream b = new DataOutputStream(a);
        rr2.serialize(b);

Connection conn = GetConnection("test", "Gauss_234"); //User name and password.
        Statement pstmt = null;
        try {
            conn.setAutoCommit(true);
            pstmt = conn.createStatement();

            pstmt.execute("drop table if exists t_rb");
            pstmt.execute("create table t_rb(c1 int, c2 roaringbitmap) distribute by hash (c1);");

            StringReader sr = null;
            CopyManager cm = null;
            cm = new CopyManager((BaseConnection) conn);

            String delimiter = "|";
            StringBuffer tuples = new StringBuffer();
            tuples.append("1" + delimiter + "\\x" + bytesToHex(a.toByteArray()));


            StringBuffer sb = new StringBuffer();
            sb.append(tuples.toString());

            sr = new StringReader(tuples.toString());
            String sql = "copy t_rb from STDIN with (delimiter '|', NOESCAPING)";

long rows = cm.copyIn(sql, sr);//Execute the COPY command to save data to the database.

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

11.2.9 JDBC Interfaces

JDBC interface is a set of API methods for users. This section describes some common interfaces. For other interfaces, see information in JDK1.6 (software package) and JDBC 4.0.

java.sql.Connection

This section describes **java.sql.Connection**, the interface for connecting to a database.

Table 11-7 java.sql.Connection methods

Method	Return Type	Support JDBC 4 or Not
close()	void	Yes
commit()	void	Yes
createStatement()	Statement	Yes
getAutoCommit()	boolean	Yes
getClientInfo()	Properties	Yes
getClientInfo(String name)	String	Yes
getTransactionIsolation()	int	Yes
isClosed()	boolean	Yes
isReadOnly()	boolean	Yes
prepareStatement(String sql)	PreparedStatement	Yes
rollback()	void	Yes
setAutoCommit(boolean autoCommit)	void	Yes
setClientInfo(Properties properties)	void	Yes
setClientInfo(String name,String value)	void	Yes

NOTICE

The interface uses the AutoCommit mode by default, but you can disable it by setting **setAutoCommit** to **false**. This will package all subsequent statements in explicit transactions. Note that you will not be able to execute statements that cannot be executed within transactions.

java.sql.CallableStatement

This section describes **java.sql.CallableStatement**, the stored procedure execution interface.

Table 11-8 java.sql.CallableStatement methods

Method	Return Type	Support JDBC 4 or Not
registerOutParameter(int parameterIndex, int type)	void	Yes
wasNull()	boolean	Yes
getString(int parameterIndex)	String	Yes
getBoolean(int parameterIndex)	boolean	Yes
getByte(int parameterIndex)	byte	Yes
getShort(int parameterIndex)	short	Yes
getInt(int parameterIndex)	int	Yes
getLong(int parameterIndex)	long	Yes
getFloat(int parameterIndex)	float	Yes
getDouble(int parameterIndex)	double	Yes
getBigDecimal(int parameterIndex)	BigDecimal	Yes
getBytes(int parameterIndex)	byte[]	Yes
getDate(int parameterIndex)	Date	Yes
getTime(int parameterIndex)	Time	Yes
getTimestamp(int parameterIndex)	Timestamp	Yes
getObject(int parameterIndex)	Object	Yes

NOTE

- Do not perform batch operations on statements containing **OUT** parameters.
- 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**.

java.sql.DatabaseMetaData

This section describes **java.sql.DatabaseMetaData**, the interface for defining database objects.

Table 11-9 java.sql.DatabaseMetaData methods

Method	Return Type	Support JDBC 4 or Not
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
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

Method	Return Type	Support JDBC 4 or Not
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
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

Method	Return Type	Support JDBC 4 or Not
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
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

java.sql.Driver

This section describes **java.sql.Driver**, the database driver interface.

Table 11-10 java.sql.Driver methods

Method	Return Type	Support JDBC 4 or Not
acceptsURL(String url)	boolean	Yes
connect(String url, Properties info)	Connection	Yes
jdbcCompliant()	boolean	Yes
getMajorVersion()	int	Yes
getMinorVersion()	int	Yes

java.sql.PreparedStatement

This section describes **java.sql.PreparedStatement**, the interface for preparing statements.

Table 11-11 java.sql.PreparedStatement methods

Method	Return Type	Support JDBC 4 or Not
clearParameters()	void	Yes
execute()	boolean	Yes
executeQuery()	ResultSet	Yes
executeUpdate()	int	Yes
getMetaData()	ResultSetMetaData	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

Method	Return Type	Support JDBC 4 or Not
setFloat(int parameterIndex, float x)	void	Yes
setInt(int parameterIndex, int x)	void	Yes
setLong(int parameterIndex, long x)	void	Yes
setNString(int parameterIndex, String value)	void	Yes
setShort(int parameterIndex, short x)	void	Yes
setString(int parameterIndex, String x)	void	Yes
addBatch()	void	Yes
executeBatch()	int[]	Yes
clearBatch()	void	Yes

NOTE

- **addBatch()** and **execute()** can be executed only after **clearBatch()**.
- Calling the **executeBatch()** method does not clear the batch. Clear batch by explicitly calling **clearBatch()**.
- You do not need to use **set*()** to reuse the values of bounded variables in a batch after they have been added.
- The following methods are inherited from **java.sql.Statement**: **close**, **execute**, **executeQuery**, **executeUpdate**, **getConnection**, **getResultSet**, **getUpdateCount**, **isClosed**, **setMaxRows**, and **setFetchSize**.

java.sql.ResultSet

This section describes **java.sql.ResultSet**, the interface for execution result sets.

Table 11-12 java.sql.ResultSet methods

Method	Return Type	Support JDBC 4 or Not
findColumn(String columnLabel)	int	Yes
getBigDecimal(int columnIndex)	BigDecimal	Yes

Method	Return Type	Support JDBC 4 or Not
getBigDecimal(String columnLabel)	BigDecimal	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
getDate(int columnIndex)	Date	Yes
getDate(String columnLabel)	Date	Yes
getDouble(int columnIndex)	double	Yes
getDouble(String columnLabel)	double	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
getShort(int columnIndex)	short	Yes
getShort(String columnLabel)	short	Yes
getString(int columnIndex)	String	Yes
getString(String columnLabel)	String	Yes

Method	Return Type	Support JDBC 4 or Not
getTime(int columnIndex)	Time	Yes
getTime(String columnLabel)	Time	Yes
getTimestamp(int columnIndex)	Timestamp	Yes
getTimestamp(String columnLabel)	Timestamp	Yes
isAfterLast()	boolean	Yes
isBeforeFirst()	boolean	Yes
isFirst()	boolean	Yes
next()	boolean	Yes

NOTE

- A statement cannot have multiple open result sets.
- The cursor used to traverse the result set cannot remain in the open state after being committed.

java.sql.ResultSetMetaData

This section describes **java.sql.ResultSetMetaData**, which provides details about ResultSet object information.

Table 11-13 java.sql.ResultSetMetaData methods

Method	Return Type	Support JDBC 4 or Not
getColumnCount()	int	Yes
getColumnName(int column)	String	Yes
getColumnType(int column)	int	Yes
getColumnTypeName(int column)	String	Yes

java.sql.Statement

This section describes **java.sql.Statement**, the interface for executing SQL statements.

Table 11-14 java.sql.Statement methods

Method	Return Type	Support JDBC 4 or Not
close()	void	Yes
execute(String sql)	boolean	Yes
executeQuery(String sql)	ResultSet	Yes
executeUpdate(String sql)	int	Yes
getConnection()	Connection	Yes
getResultSet()	ResultSet	Yes
getQueryTimeout()	int	Yes
getUpdateCount()	int	Yes
isClosed()	boolean	Yes
setQueryTimeout(int seconds)	void	Yes
setFetchSize(int rows)	void	Yes
cancel()	void	Yes

NOTE

setFetchSize can reduce the memory occupied by the result set 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. Therefore, when setting **setFetchSize**, set **setAutoCommit** to **false** and commit transactions on the connection to flush service data to a database.

javax.sql.ConnectionPoolDataSource

This section describes **javax.sql.ConnectionPoolDataSource**, the interface for data source connection pools.

Table 11-15 javax.sql.ConnectionPoolDataSource methods

Method	Return Type	Support JDBC 4 or Not
getLoginTimeout()	int	Yes
getLogWriter()	PrintWriter	Yes
getPooledConnection()	PooledConnection	Yes

Method	Return Type	Support JDBC 4 or Not
getPooledConnection(String user,String password)	PooledConnection	Yes
setLoginTimeout(int seconds)	void	Yes
setLogWriter(PrintWriter out)	void	Yes

javax.sql.DataSource

This section describes **javax.sql.DataSource**, the interface for data sources.

Table 11-16 javax.sql.DataSource methods

Method	Return Type	Support JDBC 4 or Not
getConnection()	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

javax.sql.PooledConnection

This section describes **javax.sql.PooledConnection**, the connection interface created by a connection pool.

Table 11-17 javax.sql.PooledConnection methods

Method	Return Type	Support JDBC 4 or Not
addConnectionEventListener (ConnectionEventListener listener)	void	Yes
close()	void	Yes
getConnection()	Connection	Yes

Method	Return Type	Support JDBC 4 or Not
removeConnectionEventListener (ConnectionEventListener listener)	void	Yes
addStatementEventListener (StatementEventListener listener)	void	Yes
removeStatementEventListener (StatementEventListener listener)	void	Yes

javax.naming.Context

This section describes **javax.naming.Context**, the context interface for connection configuration.

Table 11-18 javax.naming.Context methods

Method	Return Type	Support JDBC 4 or Not
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

javax.naming.spi.InitialContextFactory

This section describes **javax.naming.spi.InitialContextFactory**, the initial context factory interface.

Table 11-19 javax.naming.spi.InitialContextFactory methods

Method	Return Type	Support JDBC 4 or Not
getInitialContext(Hashtable<?,?> environment)	Context	Yes

CopyManager

CopyManager is an API interface class provided by the JDBC driver in GaussDB(DWS). It is used to import data to GaussDB(DWS) in batches.

Inheritance relationship of CopyManager

The CopyManager class is in the **org.postgresql.copy** package class 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 11-20 Common methods of CopyManager

Returned 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
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

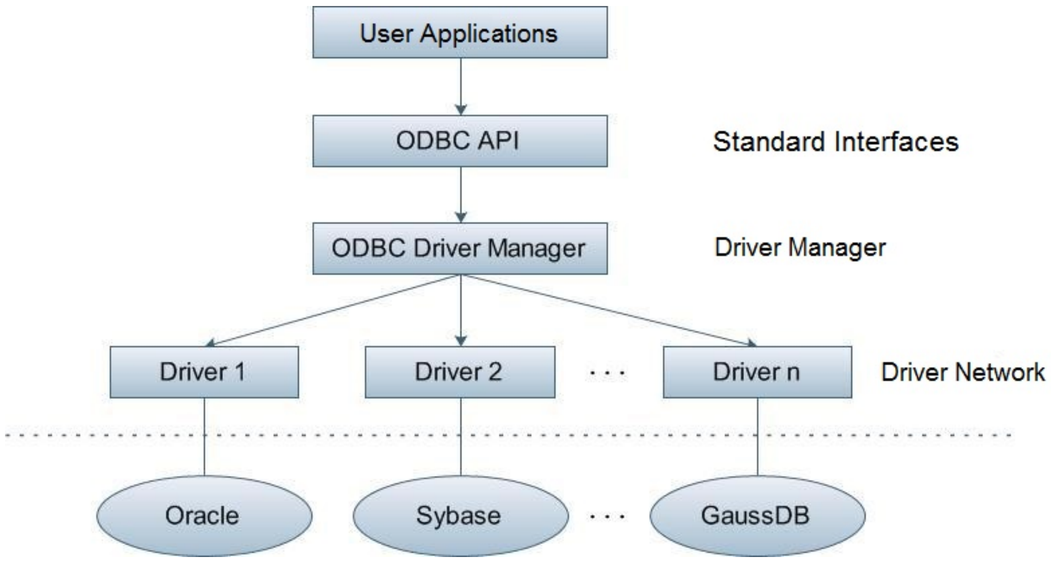
Returned Value	Method	Description	throws
long	copyIn(String sql, Reader from)	Uses **COPY FROM STDIN** to quickly load data to tables in the database from Reader.	SQLException,IOException
long	copyIn(String sql, Reader from, int bufferSize)	Uses **COPY FROM STDIN** to quickly load data to tables in the 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

11.3 ODBC-Based Development

Open Database Connectivity (ODBC) is an MS API for accessing databases based on the X/OPEN CLI. The ODBC API alleviates applications from directly operating in databases, and enhances the database portability, extensibility, and maintainability.

Figure 11-2 shows the system structure of ODBC.

Figure 11-2 ODBC system structure



GaussDB(DWS) supports ODBC 3.5 in the following environments.

Table 11-21 OSs Supported by ODBC

OS	Platform
SUSE Linux Enterprise Server 11 SP1/SP2/SP3/SP4 SUSE Linux Enterprise Server 12 and SP1/SP2/SP3/SP5	x86_64
Red Hat Enterprise Linux 6.4/6.5/6.6/6.7/6.8/6.9/7.0/7.1/7.2/7.3/7.4/7.5	x86_64
Red Hat Enterprise Linux 7.5	ARM64
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
NeoKylin 7.5/7.6	ARM64
Oracle Linux R7U4	x86_64
Windows 7	32-bit
Windows 7	64-bit
Windows Server 2008	32-bit
Windows Server 2008	64-bit

The operating systems listed above refer to the operating systems on which the ODBC program runs. They can be different from the operating systems where databases are deployed.

The ODBC Driver Manager running on UNIX or Linux can be unixODBC or iODBC. Select unixODBC-2.3.0 here as the component for connecting 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 GaussDB(DWS) data types, such as tinyint, smalldatetime, and nvarchar2.

11.3.1 ODBC Package and Its Dependent Libraries and Header Files

Download the ODBC software package from the console.

For details, see **Downloading the JDBC or ODBC Driver**.

ODBC Package for the Linux OS

Obtain the **dws_8.1.*x*_odbc_driver_for_*xxx*_*xxx*.zip** package from the software package. In the Linux OS, header files (including **sql.h** and **sqlext.h**) and library (**libodbc.so**) are required in application development. These header files and libraries can be obtained from the unixODBC-2.3.0 installation package.

ODBC Package for the Windows OS

Obtain the **dws_8.1.*x*_odbc_driver_for_windows.zip** package from the software package. In the Windows OS, the required header files and library files are system-resident.

11.3.2 Configuring a Data Source in the Linux OS

The ODBC DRIVER (psqlodbcw.so) provided by GaussDB(DWS) can be used after it has been configured in the data source. To configure data sources, users 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 at: Currently, unixODBC-2.2.1 is not supported. unixODBC-2.3.0 is used as an example.

https://sourceforge.net/projects/unixodbc/files/unixODBC/2.3.0/unixODBC-2.3.0.tar.gz/download

Step 2 Prepare **unixODBC**.

1. Decompress the **unixODBC** code file.

```
tar -xvf unixODBC-2.3.0.tar.gz
```

2. Compile the code file and install the driver.

```
cd unixODBC-2.3.0
./configure --enable-gui=no
make
make install
```

NOTE

- After the unixODBC is compiled and installed, the ***.so.2** library file will be in the installation directory. To create the ***.so.1** library file, change **LIB_VERSION** in the configure file to **1:0:0**.
  ```
  LIB_VERSION="1:0:0"
  ```
- This driver dynamically loads the **libodbcinst.so.*** library files. If one of the library files is successfully loaded, the library file is loaded. The loading priority is **libodbcinst.so** > **libodbcinst.so.1** > **libodbcinst.so.1.0.0** > **libodbcinst.so.2** > **libodbcinst.so.2.0.0**.

  For example, a directory can be dynamically linked to **libodbcinst.so.1**, **libodbcinst.so.1.0.0**, and **libodbcinst.so.2**. The driver file loads **libodbcinst.so** first. If **libodbcinst.so** cannot be found in the current environment, the driver file searches for **libodbcinst.so.1**, which has a lower priority. After **libodbcinst.so.1** is loaded, the loading is complete.

Step 3 Replace the GaussDB(DWS) client driver.

Decompress **dws_8.1.***x***_odbc_driver_for_***xxx_xxx***.zip** to obtain the **psqlodbcw.la** and **psqlodbcw.so** files in the **/dws_8.1.***x***_odbc_driver_for_***xxx_xxx***/odbc/lib** directory.

Step 4 Configure the data source.

1. Configure the ODBC driver file.

   Add the following content to the end of 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 11-22**.

Table 11-22 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=/xxx/odbc/lib/psqlodbcw.so
setup	Driver installation path, which is the same as the dynamic library path in Driver64.	setup=/xxx/odbc/lib/psqlodbcw.so

2. Configure the data source file.

   Add the following content to the end of the **/usr/local/etc/odbc.ini** file:

```
[MPPODBC]
Driver=GaussMPP
Servername=10.10.0.13 (database server IP address)
Database=gaussdb (database name)
```

```
Username=dbadmin (database username)
Password= (database user password)
Port=8000 (database listening port)
Sslmode=allow
```

For descriptions of the parameters in the **odbc.ini** file, see **Table 11-23**.

Table 11-23 odbc.ini configuration parameters

Parameter	Description	Example
[DSN]	Data source name	[MPPODBC]
Driver	Driver name, corresponding to DriverName in **odbcinst.ini**	Driver=DRIVER_N
Servername	IP address of the server	Servername=10.145.130.26
Database	Name of the database to connect to	Database=gaussdb
Username	Name of the database user	Username=dbadmin
Password	Password of the database user	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 ID of the server	Port=8000
Sslmode	Whether to enable the SSL mode	Sslmode=allow

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, the 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 the **support_batch_bind** parameter is set to **on**, the batch query protocol is enabled.	UseBatchProtocol=1
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 **1**. If this parameter is set to **0**, the ODBC driver reports the name and version of the current driver to the database. If this parameter is set to **1**, the ODBC driver reports the name, deployment path, and process owner of the current driver to the database and records them in the **connection_info** parameter (see **connection_info**). You can query this parameter in **PG_STAT_ACTIVITY** and **PGXC_STAT_ACTIVITY**.

Parameter	Description	Example
ForExtensionConnector	ETL tool performance optimization parameter. It can be used to optimize the memory and reduce the memory usage by the peer CN, to avoid system instability caused by excessive CN memory usage. The value can be **0** or **1**. The default value is **0**, indicating that the optimization item is disabled. Do not set this parameter for other services outside the database system. Otherwise, the service correctness may be affected.	ForExtensionConnector=1

KeepDisallowPremature	Specifies whether the cursor in the SQL statement has the **with hold** attribute when the following conditions are met: **UseDeclareFetch** is set to **1**, and the application invokes **SQLNumResultCols**, **SQLDescribeCol**, or **SQLColAttribute** after invoking **SQLPrepare** to obtain the column information of the result set.
	The value can be **0** or **1**. **0** indicates that the with hold attribute is supported, and **1** indicates that the with hold attribute is not supported. The default value is **0**.
		KeepDisallowPremature=1
		NOTE
		When **UseServerSidePrepare** is set to **1**, the **KeepDisallowPremature** parameter does not take effect. To use this parameter, set **UseServerSidePrepare** to **0**. For example, set **UseDeclareFetch** to **1**.
		KeepDisallowPremature=1
		UseServerSidePrepare=0

The valid values of **sslmode** are as follows.

Table 11-24 sslmode options

sslmode	Whether SSL Encryption Is Enabled	Description
disable	No	The SSL secure connection is not used.
allow	Probably	The SSL secure encrypted connection is used if required by the database server, but does not check the authenticity of the server.
prefer	Probably	The SSL secure encrypted connection is used as a preferred mode if supported by the database, but does not check the authenticity of the server.
require	Yes	The SSL secure connection must be used, but it only encrypts data and does not check the authenticity of the server.
verify-ca	Yes	The SSL secure connection must be used, and it checks whether the database has certificates issued by a trusted CA.
verify-full	Yes	The SSL secure connection must be used. In addition to the check scope specified by **verify-ca**, it checks whether the name of the host where the database resides is the same as that on the certificate. This mode is not supported.

Step 5 Enable the SSL mode.

To use SSL certificates for connection, decompress the certificate package contained in the GaussDB(DWS) installation package, and run **source sslcert_env.sh** in a shell environment to deploy certificates in the default location of the current session.

Or manually declare the following environment variables and ensure that the permission for the client.key* series files is set to 600.

```
export PGSSLCERT= "/YOUR/PATH/OF/client.crt" # Change the path to the absolute path of client.crt.
export PGSSLKEY= "/YOUR/PATH/OF/client.key" # Change the path to the absolute path of client.key.
```

In addition, change the value of **Sslmode** in the data source to **verify-ca**.

Step 6 Add the IP address segment of the host where the client is located to the security group rules of GaussDB(DWS) to ensure that the host can communicate with GaussDB(DWS).

Step 7 Configure environment variables.

```
vim ~/.bashrc
```

Add the following content to the end of 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 8 Run the following commands to validate the settings:

```
source ~/.bashrc
```

----End

Testing Data Source Configuration

Run the **isql** -*v GaussODBC* command (*GaussODBC* 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][Driver Manager]Can't open lib 'xxx/xxx/psqlodbcw.so' : file not found.

  Possible causes:

  – The path configured in the **odbcinst.ini** file is incorrect.

    Run **ls** to check the path in the error information, ensuring that the **psqlodbcw.so** file exists and you have execution permissions on it.

  – The dependent library of **psqlodbcw.so** does not exist or is not in system environment variables.

    Run **ldd** to check the path in the error information. If **libodbc.so.1** or other UnixODBC libraries are lacking, configure UnixODBC again following the procedure provided in this section, and add the **lib** directory under its installation directory to **LD_LIBRARY_PATH**. If other libraries are lacking, add the **lib** directory under the ODBC driver package to **LD_LIBRARY_PATH**.

- [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 network adapter and port are monitored based on database server configurations in the procedure in this section.

  – Firewall and network gatekeeper settings are improper.

    Check firewall settings, ensuring 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 it to **allow** or a higher level. For more details, see **Table 11-24**.

- 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.

- FATAL: Forbid remote connection with trust method!

  For security purposes, the CN forbids access from other nodes in the cluster without authentication.

  To access the CN from inside the cluster, deploy the ODBC program on the machine where the CN is located and use 127.0.0.1 as the server address. It is recommended that the service system be deployed outside the cluster. If it is deployed inside, the database performance may be affected.

- [unixODBC][Driver Manager]Invalid attribute value

  This problem occurs when you use SQL on other GaussDB. The possible cause is that the unixODBC version is not the recommended one. You are advised to run the **odbcinst --version** command to check the unixODBC version.

- 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.
- In versions earlier than V100R002C80SPC300, the database stores only SHA-256 hashes and no MD5 hashes. Therefore, MD5 cannot be used for user password authentication.
- In V100R002C80SPC300 and later, 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. If your database is upgraded from a version earlier than V100R002C80SPC300, passwords in the old version will only have SHA-256 hashes and not support MD5 authentication.

To solve this problem, you can update the user password. Alternatively, create a user, assign the same permissions to the user, and use the new user to connect to the database.

- 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.

11.3.3 Configuring a Data Source in the Windows OS

Configure the ODBC data source using the ODBC data source manager preinstalled in the Windows OS.

Procedure

Step 1 Replace the GaussDB(DWS) client driver.

Decompress **GaussDB-8.1.3-Windows-Odbc.tar.gz** and install **psqlodbc.msi** (for 32-bit OS) or **psqlodbc_x64.msi** (for 64-bit OS).

Step 2 Open Driver Manager.

Use the Driver Manager suitable for your OS to configure the data source. (Assume the Windows system drive is drive C.)

- If you develop 32-bit programs in the 64-bit Windows OS, open the 32-bit Driver Manager at **C:\Windows\SysWOW64\odbcad32.exe** after you install the 32-bit driver.

  Do not open Driver Manager by choosing **Control Panel**, clicking **Administrative Tools**, and clicking **Data Sources (ODBC)**.

  NOTE

  WoW64 is the acronym for "Windows 32-bit on Windows 64-bit". **C:\Windows\SysWOW64\** stores the 32-bit environment on a 64-bit system.

- If you develop 64-bit programs in the 64-bit Windows OS, open the 64-bit Driver Manager at **C:\Windows\System32\odbcad32.exe** after you install the 64-bit driver.

  Do not open Driver Manager by choosing **Control Panel**, clicking **Administrative Tools**, and clicking **Data Sources (ODBC)**.

NOTE

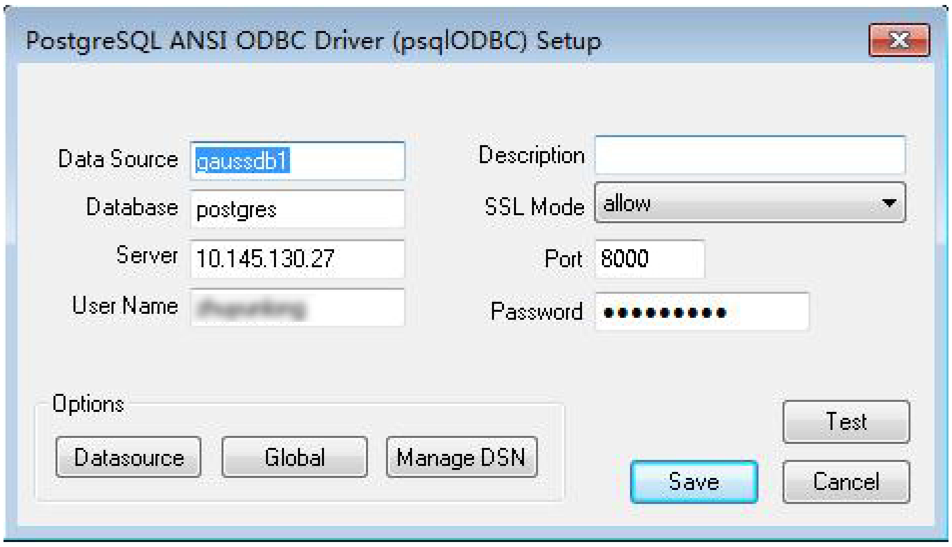
C:\Windows\System32\ stores the environment consistent with the current OS. For technical details, see Windows technical documents.

- In a 32-bit Windows OS, open **C:\Windows\System32\odbcad32.exe**.

  In the Windows OS, click **Computer**, and choose **Control Panel**. Click **Administrative Tools** and click **Data Sources (ODBC)**.

Step 3 Configure the data source.

On the **User DSN** tab, click **Add**, and choose **PostgreSQL Unicode** for setup. (An identifier will be displayed for the 64-bit OS.)



Data Source: gaussdb1
Database: postgres
Server: 10.145.130.27
User Name: [redacted]
Description:
SSL Mode: allow
Port: 8000
Password: •••••••••

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.

To use SSL certificates for connection, decompress the certificate package contained in the GaussDB(DWS) installation package, and double-click the **sslcert_env.bat** file to deploy certificates in the default location.

NOTICE

The **sslcert_env.bat** file ensures the purity of the certificate environment. When the **%APPDATA%\postgresql** directory exists, a message will be prompted asking you whether you want to remove related directories. If you want to remove related directories, back up files in the directory.

Alternatively, you can copy the **client.crt**, **client.key**, **client.key.cipher**, and **client.key.rand** files in the certificate file folder to the manually created **%APPDATA%\postgresql** directory. 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**.

Change the value of **SSL Mode** in step 2 to **verify-ca**.

Table 11-25 sslmode options

sslmode	Whether SSL Encryption Is Enabled	Description
disable	No	The SSL secure connection is not used.
allow	Probably	The SSL secure encrypted connection is used if required by the database server, but does not check the authenticity of the server.
prefer	Probably	The SSL secure encrypted connection is used as a preferred mode if supported by the database, but does not check the authenticity of the server.
require	Yes	The SSL secure connection must be used, but it only encrypts data and does not check the authenticity of the server.
verify-ca	Yes	The SSL secure connection must be used, and it checks whether the database has certificates issued by a trusted CA.
verify-full	Yes	The SSL secure connection must be used. In addition to the check scope specified by **verify-ca**, it checks whether the name of the host where the database resides is the same as that on the certificate. **NOTE** This mode cannot be used.

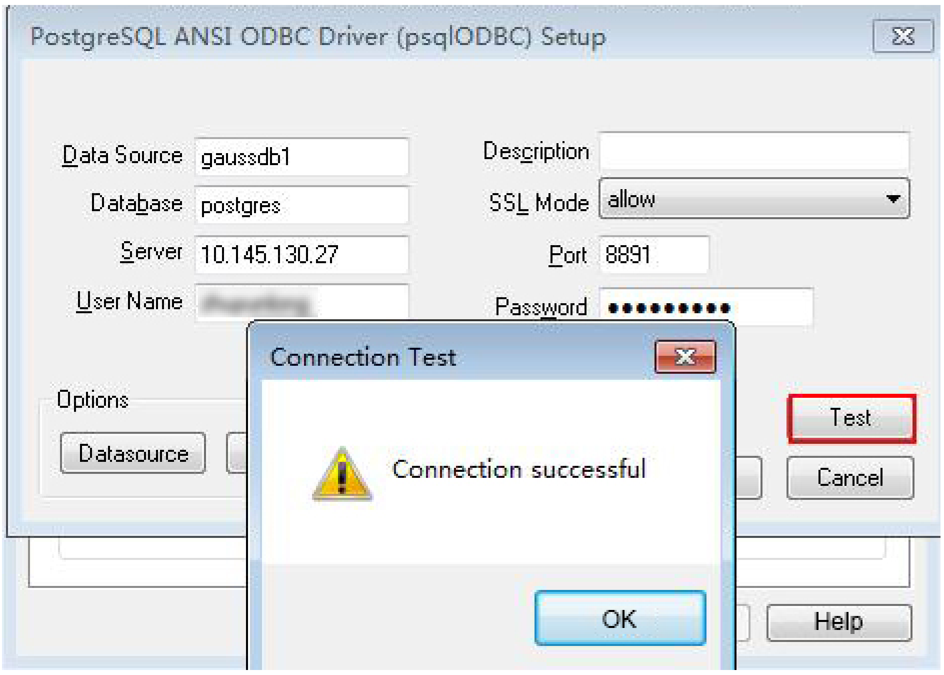
Step 5 Add the IP address segment of the host where the client is located to the security group rules of GaussDB(DWS) to ensure that the host can communicate with GaussDB(DWS).

----End

Testing Data Source Configuration

Click **Test**.

- If the following information is displayed, the configuration is correct and the connection succeeds.



Data Source gaussdb1
Database postgres
Server 10.145.130.27
User Name [redacted]
Description
SSL Mode allow
Port 8891
Password ●●●●●●●●●

Connection successful

- If error information is displayed, the configuration is incorrect. Check the configuration.

Troubleshooting

- Server common name "xxxx" does not match host name "xxxxx"

  This problem occurs because when **verify-full** is used for SSL encryption, the driver checks whether the host name in certificates is the same as the actual one. To solve this problem, use **verify-ca** to stop checking host names, or generate a set of CA certificates containing the actual host names.

- 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 network adapter and port are monitored based on database server configurations in the procedure in this section.

  – Firewall and network gatekeeper settings are improper.

    Check firewall settings, ensuring that the database communication port is trusted.

    Check to ensure network gatekeeper settings are proper (if any).

- In the specified DSN, the system structures of the drive do not match those of the application.

  Possible cause: The bit versions of the drive and program are different.

  C:\Windows\SysWOW64\odbcad32.exe is a 32-bit ODBC Drive Manager.

  C:\Windows\System32\odbcad32.exe is a 64-bit ODBC Drive Manager.

- 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 connections. For details about **sslmode**, see **Table 11-25**.

- 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.
  - In versions earlier than V100R002C80SPC300, the database stores only SHA-256 hashes and no MD5 hashes. Therefore, MD5 cannot be used for user password authentication.
  - In V100R002C80SPC300 and later, 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. If your database is upgraded from a version earlier than V100R002C80SPC300, passwords in the old version will only have SHA-256 hashes and not support MD5 authentication.

  To solve this problem, perform the following operations:

  a. Set **password_encryption_type** to **1**. For details, see **Modifying Database Parameters**.

  b. Create a new database user for connection or reset the password of the existing database user.

     ■ If you use an administrator account, reset the password. For details, see **Resetting a Password**.

     ■ If you are a common user, use another client tool (such as Data Studio) to connect to the database and run the **ALTER USER** statement to change your password.

  c. Connect to the database.

- 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.

11.3.4 ODBC Development Example

Code for Common Functions

The following example shows how to obtain data from GaussDB(DWS) 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;
SQLLEN         V_StrLen_or_IndPtr;
int main(int argc,char *argv[])
{
      // 1. Apply for 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. Apply for 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. Apply for a statement handle.
      SQLAllocHandle(SQL_HANDLE_STMT, V_OD_hdbc, &V_OD_hstmt);
      // 8. Execute an SQL statement directly.
      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 the 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. Execute the prepared statement.
      SQLExecute(V_OD_hstmt);
      SQLExecDirect(V_OD_hstmt,"select id from testtable",SQL_NTS);
      // 12. Obtain the attributes of a certain column in the result set.

SQLColAttribute(V_OD_hstmt,1,SQL_DESC_TYPE_NAME,typename,sizeof(typename),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_StrLen_or_IndPtr);
      // 14. Collect data using SQLFetch.
      V_OD_erg=SQLFetch(V_OD_hstmt);
      // 15. Obtain and return data using 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 from the data source 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 **support_batch_bind** to **on**.
- Use **CHECK_ERROR** 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);

    retcode = SQLExecute(hstmt);
    sprintf((char*)loginfo, "SQLExecute stmt log: %s", (char*)sql);
    CHECK_ERROR(retcode, loginfo, hstmt, SQL_HANDLE_STMT);

    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 amount of data to be batch processed.
    getInt("batchCount", &batchCount, 'N', 1);
    do
    {
// Interactively obtain the amount 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 Version
    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. 
That is, 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);
        }

        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;
}
```

11.3.5 ODBC Interfaces

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).

SQLAllocEnv

In ODBC 3.*x*, **SQLAllocEnv** (a function in ODBC 2.*x*) was deprecated and replaced by **SQLAllocHandle**. For details, see **SQLAllocHandle**.

SQLAllocConnect

In ODBC 3.*x*, **SQLAllocConnect** (a function in ODBC 2.*x*) was deprecated and replaced by **SQLAllocHandle**. For details, see **SQLAllocHandle**.

SQLAllocHandle

Function

SQLAllocHandle allocates environment, connection, or statement handles. It replaces the ODBC 2.*x* functions **SQLAllocEnv**, **SQLAllocConnect**, and **SQLAllocStmt**.

Prototype

```
SQLRETURN SQLAllocHandle(SQLSMALLINT   HandleType,
                         SQLHANDLE     InputHandle,
                         SQLHANDLE     *OutputHandlePtr);
```

Parameters

Table 11-26 SQLAllocHandle parameters

Parameter	Description
HandleType	Handle type 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** (description 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.

Parameter	Description
InputHandle	Type of the new handle to be allocated. ● If **HandleType** is **SQL_HANDLE_ENV**, the value is **SQL_NULL_HANDLE**. ● If **HandleType** is **SQL_HANDLE_DBC**, this must be an environment handle. ● If **HandleType** is **SQL_HANDLE_STMT** or **SQL_HANDLE_DESC**, it must be a connection handle.
OutputHandlePtr	**Output parameter**: Pointer to a buffer in which the handle returned for the newly allocated data structure is stored.

Return values

- **SQL_SUCCESS** indicates that the call is successful.
- **SQL_SUCCESS_WITH_INFO** indicates warning information.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection setup failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. Values returned by other APIs are similar to the values returned by the API you have used.

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**, set **HandleType** and **Handle** to **IntputHandle**, and obtain the **SQLSTATE** value. This value can be used to get more information about the function call.

Examples

See **ODBC Development Example**.

SQLAllocStmt

In ODBC 3.*x*, **SQLAllocStmt** (a function in ODBC 2.*x*) was deprecated and replaced by **SQLAllocHandle**. For details, see **SQLAllocHandle**.

SQLBindCol

Function

SQLBindCol is used to associate (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 11-27 SQLBindCol parameters

Parameter	Description
StatementHandle	Statement handle.
ColumnNumber	Number of the column to be bound. Column numbering begins at 0 and increases in ascending order. Column **0** functions as the bookmark. If no bookmark column is set, column numbering begins at 1 instead.
TargetType	The 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.
BufferLength	Length of the buffer to which **TargetValuePtr** points, 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 values

- **SQL_SUCCESS** indicates that the call is successful.
- **SQL_SUCCESS_WITH_INFO** indicates warning information.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection setup failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. Values returned by other APIs are similar to the values returned by the API you have used.

Note

If **SQLBindCol** returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can then call **SQLGetDiagRec**, set **HandleType** and **Handle** to **SQL_HANDLE_STMT** and **StatementHandle**, and obtain the **SQLSTATE** value. This value can be used to get more information about the function call.

Examples

See **ODBC Development Example**.

SQLBindParameter

Function

SQLBindParameter binds a parameter flag 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 11-28 SQLBindParameter

Keyword	Description
StatementHandle	Statement handle.
ParameterNumber	Parameter marker number, starting at 1 and increasing in an ascending order.
InputOutputType	Input and output parameter types.
ValueType	C data type of the parameter.
ParameterType	SQL data type of the parameter.
ColumnSize	Column size or the expression of the corresponding parameter marker.
DecimalDigits	Decimal number of the column or the expression of the corresponding parameter marker.
ParameterValuePtr	Pointer to the buffer for storing parameter data.
BufferLength	Length of the buffer to which the **ParameterValuePtr** points, 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 values

- **SQL_SUCCESS** indicates that the call is successful.
- **SQL_SUCCESS_WITH_INFO** indicates warning information.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection setup failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. Values returned by other APIs are similar to the values returned by the API you have used.

Precautions

If **SQLBindCol** returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can then call **SQLGetDiagRec**, set **HandleType** and **Handle** to **SQL_HANDLE_STMT** and **StatementHandle**, and obtain the **SQLSTATE** value. This value can be used to get more information about the function call.

Examples

See **ODBC Development Example**.

SQLColAttribute

Function

SQLColAttribute returns the descriptor information about a column in the result set.

Prototype

```
SQLRETURN SQLColAttribute(SQLHSTMT        StatementHandle,
                          SQLUSMALLINT    ColumnNumber,
                          SQLUSMALLINT    FieldIdentifier,
                          SQLPOINTER      CharacterAtrriburePtr,
                          SQLSMALLINT     BufferLength,
                          SQLSMALLINT     *StringLengthPtr,
                          SQLPOINTER      NumericAttributePtr);
```

Parameters

Table 11-29 SQLColAttribute parameters

Parameter	Description
StatementHandle	Statement handle.
ColumnNumber	Column number of the field to be queried, starting at 1 and increasing in an ascending order.
FieldIdentifier	Field identifier of **ColumnNumber** in IRD.
CharacterAttributePtr	**Output parameter**: pointer to the buffer that returns FieldIdentifier field value.
BufferLength	● **FieldIdentifier** indicates the buffer length when it refers to an ODBC-defined field and **CharacterAttributePtr** points to a string or 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 the **FieldIdentifier** field in the **ColumnNumber** row of the IRD is returned.

Return values

- **SQL_SUCCESS** indicates that the call is successful.
- **SQL_SUCCESS_WITH_INFO** indicates warning information.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection setup failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. Values returned by other APIs are similar to the values returned by the API you have used.

Precautions

If **SQLColAttribute** returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can then call **SQLGetDiagRec**, set **HandleType** and **Handle** to **SQL_HANDLE_STMT** and **StatementHandle**, and obtain the **SQLSTATE** value. This value can be used to get more information about the function call.

Examples

See **ODBC Development Example**.

SQLConnect

Function

SQLConnect establishes a connection between a driver and a data source. Using the connection handle, you can obtain crucial information like the program's status, transaction processing status, and error messages after establishing a connection to the data source.

Prototype

```
SQLRETURN  SQLConnect(SQLHDBC        ConnectionHandle,
                      SQLCHAR        *ServerName,
                      SQLSMALLINT    NameLength1,
                      SQLCHAR        *UserName,
                      SQLSMALLINT    NameLength2,
                      SQLCHAR        *Authentication,
                      SQLSMALLINT    NameLength3);
```

Parameters

Table 11-30 SQLConnect parameters

Parameter	Description
ConnectionHandle	Connection handle, obtained from **SQLAllocHandle**.
ServerName	Name of the data source to connect to.
NameLength1	Length of **ServerName**.
UserName	Database username in the data source.
NameLength2	Length of **UserName**.
Authentication	Password of the database user in the data source.

Parameter	Description
NameLength3	Length of **Authentication**.

Return values

- **SQL_SUCCESS** indicates that the call is successful.
- **SQL_SUCCESS_WITH_INFO** indicates warning information.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection setup failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. Values returned by other APIs are similar to the values returned by the API you have used.
- **SQL_STILL_EXECUTING** indicates that the statement is being executed.

Precautions

If **SQLConnect** returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can then call **SQLGetDiagRec**, set **HandleType** and **Handle** to **SQL_HANDLE_DBC** and **ConnectionHandle**, and obtain the **SQLSTATE** value. This value can be used to get more information about the function call.

Examples

See **ODBC Development Example**.

SQLDisconnect

Function

SQLDisconnect closes the connection associated with the database connection handle.

Prototype

```
SQLRETURN SQLDisconnect(SQLHDBC    ConnectionHandle);
```

Parameters

Table 11-31 SQLDisconnect parameters

Parameter	Description
ConnectionHandle	Connection handle, obtained from **SQLAllocHandle**.

Return values

- **SQL_SUCCESS** indicates that the call is successful.
- **SQL_SUCCESS_WITH_INFO** indicates warning information.

- **SQL_ERROR** indicates major errors, such as memory allocation and connection setup failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. Values returned by other APIs are similar to the values returned by the API you have used.

Precautions

If **SQLDisconnect** returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can then call **SQLGetDiagRec**, set **HandleType** and **Handle** to **SQL_HANDLE_DBC** and **ConnectionHandle**, and obtain the **SQLSTATE** value. This value can be used to get more information about the function call.

Examples

See **ODBC Development Example**.

SQLExecDirect

Function

SQLExecDirect executes a prepared SQL statement specified in this parameter. This is the fastest execution method for executing only one SQL statement at a time.

Prototype

```
SQLRETURN SQLExecDirect(SQLHSTMT         StatementHandle,
                        SQLCHAR          *StatementText,
                        SQLINTEGER       TextLength);
```

Parameters

Table 11-32 SQLExecDirect parameters

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 values

- **SQL_SUCCESS** indicates that the call is successful.
- **SQL_SUCCESS_WITH_INFO** indicates warning information.
- **SQL_NEED_DATA** indicates that there are not enough parameters provided to execute the SQL statement.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection setup failures.

- **SQL_INVALID_HANDLE** indicates that invalid handles were called. Values returned by other APIs are similar to the values returned by the API you have used.
- **SQL_STILL_EXECUTING** indicates that the statement is being executed.
- **SQL_NO_DATA** indicates that no result set is returned for the SQL statement.

Precautions

If **SQLExecDirect** returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can then call **SQLGetDiagRec**, set **HandleType** and **Handle** to **SQL_HANDLE_STMT** and **StatementHandle**, and obtain the **SQLSTATE** value. This value can be used to get more information about the function call.

Examples

See **ODBC Development Example**.

SQLExecute

Function

When a statement includes a parameter marker, the **SQLExecute** function executes a prepared SQL statement using the current value of the marker.

Prototype

```
SQLRETURN SQLExecute(SQLHSTMT    StatementHandle);
```

Parameters

Table 11-33 SQLExecute parameters

Parameter	Description
StatementHandle	Statement handle to be executed.

Return values

- **SQL_SUCCESS** indicates that the call is successful.
- **SQL_SUCCESS_WITH_INFO** indicates warning information.
- **SQL_NEED_DATA** indicates that there are not enough parameters provided to execute the SQL statement.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection setup failures.
- **SQL_NO_DATA** indicates that no result set is returned for the SQL statement.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. Values returned by other APIs are similar to the values returned by the API you have used.
- **SQL_STILL_EXECUTING** indicates that the statement is being executed.

Precautions

If **SQLExecute** returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can then call **SQLGetDiagRec**, set **HandleType** and **Handle** to **SQL_HANDLE_STMT** and **StatementHandle**, and obtain the **SQLSTATE** value. This value can be used to get more information about the function call.

Examples

See **ODBC Development Example**.

SQLFetch

Function

SQLFetch advances the cursor to the next row of the result set and retrieves any bound columns.

Prototype

```
SQLRETURN SQLFetch(SQLHSTMT    StatementHandle);
```

Parameters

Table 11-34 SQLFetch parameters

Parameter	Description
StatementHandle	Statement handle, obtained from **SQLAllocHandle**.

Return values

- **SQL_SUCCESS** indicates that the call is successful.
- **SQL_SUCCESS_WITH_INFO** indicates warning information.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection setup failures.
- **SQL_NO_DATA** indicates that no result set is returned for the SQL statement.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. Values returned by other APIs are similar to the values returned by the API you have used.
- **SQL_STILL_EXECUTING** indicates that the statement is being executed.

Precautions

If **SQLFetch** returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can then call **SQLGetDiagRec**, set **HandleType** and **Handle** to **SQL_HANDLE_STMT** and **StatementHandle**, and obtain the **SQLSTATE** value. This value can be used to get more information about the function call.

Examples

See **ODBC Development Example**.

SQLFreeStmt

In ODBC 3.*x*, **SQLFreeStmt** (a function in ODBC 2.*x*) was deprecated and replaced with **SQLFreeHandle**. For details, see **SQLFreeHandle**.

SQLFreeConnect

In ODBC 3.*x*, **SQLFreeConnect** (a function in ODBC 2.*x*) was deprecated and replaced with **SQLFreeHandle**. For details, see **SQLFreeHandle**.

SQLFreeHandle

Function

SQLFreeHandle releases 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);
```

Parameters

Table 11-35 SQLFreeHandle parameters

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 these values, **SQLFreeHandle** returns **SQL_INVALID_HANDLE**.
Handle	Handle to be released.

Return values

- **SQL_SUCCESS** indicates that the call is successful.
- **SQL_SUCCESS_WITH_INFO** indicates warning information.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection setup failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. Values returned by other APIs are similar to the values returned by the API you have used.

Precautions

If **SQLFreeHandle** returns **SQL_ERROR**, the handle is still valid.

Examples

See **ODBC Development Example**.

SQLFreeEnv

In ODBC 3.*x*, **SQLFreeEnv** (a function in ODBC 2.*x*) was deprecated and replaced with **SQLFreeHandle**. For details, see **SQLFreeHandle**.

SQLPrepare

Function

SQLPrepare prepares an SQL statement to be executed.

Prototype

```
SQLRETURN SQLPrepare(SQLHSTMT      StatementHandle,
                     SQLCHAR       *StatementText,
                     SQLINTEGER    TextLength);
```

Parameters

Table 11-36 SQLPrepare parameters

Parameter	Description
StatementHandle	Statement handle.
StatementText	SQL text string.
TextLength	Length of **StatementText**.

Return values

- **SQL_SUCCESS** indicates that the call is successful.
- **SQL_SUCCESS_WITH_INFO** indicates warning information.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection setup failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. Values returned by other APIs are similar to the values returned by the API you have used.
- **SQL_STILL_EXECUTING** indicates that the statement is being executed.

Precautions

If **SQLPrepare** returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can then call **SQLGetDiagRec**, set **HandleType** and **Handle** to **SQL_HANDLE_STMT** and **StatementHandle**, and obtain the **SQLSTATE** value. This value can be used to get more information about the function call.

Examples

See **ODBC Development Example**.

SQLGetData

Function

SQLGetData retrieves data for a single column in the current row of the result set. It can be called multiple 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);
```

Parameters

Table 11-37 SQLGetData parameters

Parameter	Description
StatementHandle	Statement handle, obtained from **SQLAllocHandle**.
Col_or_Param_Num	Column number of the data to be returned. The columns in the result set are numbered from 1 in ascending order. The number of the bookmark column is 0.
TargetType	Type identifier of the 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.

- **SQL_SUCCESS** indicates that the call is successful.
- **SQL_SUCCESS_WITH_INFO** indicates warning information.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection setup failures.
- **SQL_NO_DATA** indicates that no result set is returned for the SQL statement.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. Values returned by other APIs are similar to the values returned by the API you have used.

- **SQL_STILL_EXECUTING** indicates that the statement is being executed.

Precautions

If **SQLFetch** returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can then call **SQLGetDiagRec**, set **HandleType** and **Handle** to **SQL_HANDLE_STMT** and **StatementHandle**, and obtain the **SQLSTATE** value. This value can be used to get more information about the function call.

Examples

See **ODBC Development Example**.

SQLGetDiagRec

Function

SQLGetDiagRec returns the current values of multiple fields of 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);
```

Parameters

Table 11-38 SQLGetDiagRec parameters

Parameter	Description
HandleType	Handle type identifier that describes the handle type required for diagnosis. The value must be one of the following: ● SQL_HANDLE_ENV ● SQL_HANDLE_DBC ● SQL_HANDLE_STMT ● SQL_HANDLE_DESC
Handle	Handle of the diagnosis data structure. Its type is indicated by HandleType. If **HandleType** is **SQL_HANDLE_ENV**, **Handle** may be shared or non-shared environment handle.
RecNumber	Status record from which the application seeks information. Status records are numbered from 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.

Parameter	Description
MessageText	Pointer to a buffer that saves text strings of diagnostic information.
BufferLength	Length of **MessageText**.
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 values

- **SQL_SUCCESS** indicates that the call is successful.
- **SQL_SUCCESS_WITH_INFO** indicates warning information.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection setup failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. Values returned by other APIs are similar to the values returned by the API you have used.

Precautions

SQLGetDiagRec does not release diagnostic records for itself. It uses the following returned values to report execution results:

- **SQL_SUCCESS**: The function successfully returns diagnostic information.
- **SQL_SUCCESS_WITH_INFO**: The ***MessageText** buffer is too small to hold the requested diagnostic message and no diagnostic records are generated.
- SQL_INVALID_HANDLE: The handle specified by **HandType** and **Handle** is invalid.
- **SQL_ERROR**: **RecNumber** is smaller than or equal to zero, or **BufferLength** is smaller than zero.

If an ODBC function returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can then call **SQLGetDiagRec** and obtain the **SQLSTATE** value. The possible **SQLSTATE** values are listed as follows:

Table 11-39 SQLSTATE values

SQLSATATE	Error	Description
HY000	General error	An error occurred for which there is no specific **SQLSTATE**.

SQLSATATE	Error	Description
HY001	Memory allocation error	The driver is unable to allocate memory required to support execution or completion of the function.
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 connection times out before the data source responds to the request.
IM001	Function not supported by the driver	A function that is not supported by the **StatementHandle** driver is called.

Examples

See **ODBC Development Example**.

SQLSetConnectAttr

Function

SQLSetConnectAttr sets connection attributes.

Prototype

```
SQLRETURN SQLSetConnectAttr(SQLHDBC      ConnectionHandle
                     SQLINTEGER    Attribute,
                     SQLPOINTER    ValuePtr,
                     SQLINTEGER    StringLength);
```

Parameters

Table 11-40 SQLSetConnectAttr parameters

Parameter	Description
StatementtHandle	Connection handle.
Attribute	Attribute to set.

Parameter	Description
ValuePtr	Pointer to the value of **Attribute**. **ValuePtr** depends on the value of **Attribute** and can be a 32-bit unsigned integer value or a null-terminated string. If **ValuePtr** parameter is driver-specific value, it may be signed integer.
StringLength	If **ValuePtr** points to a string or a binary buffer, this parameter should be the length of ***ValuePtr**. If **ValuePtr** points to an integer, **StringLength** is ignored.

Return values

- **SQL_SUCCESS** indicates that the call is successful.
- **SQL_SUCCESS_WITH_INFO** indicates warning information.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection setup failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. Values returned by other APIs are similar to the values returned by the API you have used.

Precautions

If **SQLSetConnectAttr** returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can then call **SQLGetDiagRec**, set **HandleType** and **Handle** to **SQL_HANDLE_DBC** and **ConnectionHandle**, and obtain the **SQLSTATE** value. This value can be used to get more information about the function call.

Examples

See **ODBC Development Example**.

SQLSetEnvAttr

Function

SQLSetEnvAttr sets environment attributes.

Prototype

```
SQLRETURN SQLSetEnvAttr(SQLHENV       EnvironmentHandle
                        SQLINTEGER    Attribute,
                        SQLPOINTER    ValuePtr,
                        SQLINTEGER    StringLength);
```

Parameters

Table 11-41 SQLSetEnvAttr parameters

Parameter	Description
EnvironmentHandle	Environment handle.

Parameter	Description
Attribute	Environment attribute to be set. Its 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 value of **Attribute**. **ValuePtr** depends on the value of **Attribute** and can be a 32-bit integer value or a null-terminated string.
StringLength	If **ValuePtr** points to a string or a binary buffer, this parameter should be the length of ***ValuePtr**. If **ValuePtr** points to an integer, **StringLength** is ignored.

Return values

- **SQL_SUCCESS** indicates that the call is successful.
- **SQL_SUCCESS_WITH_INFO** indicates warning information.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection setup failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. Values returned by other APIs are similar to the values returned by the API you have used.

Precautions

If **SQLSetEnvAttr** returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can then call **SQLGetDiagRec**, set **HandleType** and **Handle** to **SQL_HANDLE_ENV** and **EnvironmentHandle**, and obtain the **SQLSTATE** value. This value can be used to get more information about the function call.

Examples

See **ODBC Development Example**.

SQLSetStmtAttr

Function

SQLSetStmtAttr sets attributes related to a statement.

Prototype

```
SQLRETURN SQLSetStmtAttr(SQLHSTMT      StatementHandle
                         SQLINTEGER    Attribute,
                         SQLPOINTER    ValuePtr,
                         SQLINTEGER    StringLength);
```

Parameters

Table 11-42 SQLSetStmtAttr parameters

Parameter	Description
StatementtHandle	Statement handle.
Attribute	Attribute to set.
ValuePtr	Pointer to the value of **Attribute**. **ValuePtr** depends on the value of **Attribute** and can be a 32-bit unsigned integer value or a pointer to a null-terminated string, a binary buffer, and a driver-specified value. If **ValuePtr** parameter is driver-specific value, it may be signed integer.
StringLength	If **ValuePtr** points to a string or a binary buffer, this parameter should be the length of ***ValuePtr**. If **ValuePtr** points to an integer, **StringLength** is ignored.

Return values

- **SQL_SUCCESS** indicates that the call is successful.
- **SQL_SUCCESS_WITH_INFO** indicates warning information.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection setup failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. Values returned by other APIs are similar to the values returned by the API you have used.

Precautions

If **SQLSetStmtAttr** returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can then call **SQLGetDiagRec**, set **HandleType** and **Handle** to **SQL_HANDLE_STMT** and **StatementHandle**, and obtain the **SQLSTATE** value. This value can be used to get more information about the function call.

Examples

See **ODBC Development Example**.

12 GaussDB(DWS) Resource Monitoring

GaussDB(DWS) provides multiple dimensional resource monitoring views to show the real-time and historical resource usage of tasks.

12.1 User Resource Monitoring

In the multi-tenant management framework, you can query the real-time or historical usage of your resources (including memory, CPU cores, storage space, temporary space, and I/Os) using the system view **PG_TOTAL_USER_RESOURCE_INFO** and the function **GS_WLM_USER_RESOURCE_INFO**, you can also query the historical usage of your resources through the system catalog **GS_WLM_USER_RESOURCE_HISTORY**.

Important Notes

- The CPU, I/O, and memory usage of all jobs on fast and slow lanes (simple jobs on fast lanes and complex jobs on slow lanes) can be monitored.
- Currently, fast lane jobs have no memory or CPU limits. They may use too many resources and go over the resource limit.
- In the DN monitoring view, I/O, memory, and CPU display the resource usage and limits of resource pools.
- In the CN monitoring view, I/O, memory, and CPU display the total resource usage and limit of all DN resource pools in the cluster.
- The DN monitoring information is updated every 5 seconds. CNs collect monitoring information from DNs every 5 seconds. Because each instance updates or collects user monitoring information independently, the monitoring information update time on each instance may be different.
- The auxiliary thread automatically invokes the persistence function every 30 seconds to make user monitoring data persistent. So, normally, you don't have to do this.
- When there are a large number of users and a large cluster, querying such real-time views will cause network latency due to the real-time communication overhead between CNs and DNs.
- Resources are not monitored for an initial administrator.

Procedure

- Query all users' resource quotas and real-time resource usage.

```
SELECT * FROM PG_TOTAL_USER_RESOURCE_INFO;
```

The result view is as follows:

```
username        | used_memory | total_memory | used_cpu | total_cpu | used_space | total_space |
used_temp_space | total_temp_space | used_spill_space | total_spill_space | read_kbytes | write_kbytes |
read_counts | write_counts | read_speed | write_speed
-----------------------+-------------+--------------+----------+-----------+------------+-------------
+-----------------+------------------+------------------+-------------------+-------------+--------------
+-------------+--------------+------------+-------------
perfadm             |           0 |            0 |        0 |         0 |          0 |          -1 |               0 |              -1
|             0 |              -1 |            0 |           0 |          0 |           0 |          0 |          0
usern               |           0 |        17250 |        0 |        48 |          0 |          -1 |               0 |              -1
|             0 |              -1 |            0 |           0 |          0 |           0 |          0 |          0
userg               |          34 |        15525 |    23.53 |        48 |          0 |          -1 |               0 |
-1 |        814955731 |              -1 |    6111952 |    1145864 |     763994 |      143233 |      42678
|       8001
userg1              |          34 |        13972 |    23.53 |        48 |          0 |          -1 |               0 |
-1 |        814972419 |              -1 |    6111952 |    1145864 |     763994 |      143233 |      42710
|       8007
(4 rows)
```

The I/O resource monitoring fields (**read_kbytes**, **write_kbytes**, **read_counts**, **write_counts**, **read_speed**, and **write_speed**) can be available only when the GUC parameter enable_user_metric_persistent is enabled.

For details about each column, see **PG_TOTAL_USER_RESOURCE_INFO**.

- Query a user's resource quota and real-time resource usage.

```
SELECT * FROM GS_WLM_USER_RESOURCE_INFO('username');
```

The query result is as follows:

```
userid | used_memory | total_memory | used_cpu | total_cpu | used_space | total_space |
used_temp_space | total_temp_space | used_spill_space | total_spill_space | read_kbytes | write_kbytes |
read_counts | write_counts | read_speed | write_speed
--------+-------------+--------------+----------+-----------+------------+-------------+-----------------
+------------------+------------------+-------------------+-------------+--------------+-------------+--------------
+------------+-------------
16407 |          18 |         1655 |        6 |        19 |   13787176 |          -1 |               0 |              -1
|             0 |              -1 |            0 |           0 |          0 |           0 |          0 |          0
(1 row)
```

- Query all users' resource quotas and historical resource usage.

```
SELECT * FROM GS_WLM_USER_RESOURCE_HISTORY;
```

The query result is as follows:

```
username        |           timestamp           | used_memory | total_memory | used_cpu | total_cpu |
used_space | total_space | used_temp_space | total_temp_space | used_spill_space | total_spill_space |
read_kbytes | write_kbytes | read_counts | write_counts | read_speed  | write_speed
-----------------------+-------------------------------+-------------+--------------+----------+-----------
+------------+-------------+-----------------+------------------+------------------+-------------------
+-------------+--------------+-------------+--------------+-------------+-------------
usern               | 2020-01-08 22:56:06.456855+08 |           0 |        17250 |        0 |        48 |          0
|          -1 |               0 |              -1 |       88349078 |              -1 |      45680 |          34 |       5710
|           8 |        320 |           0
userg               | 2020-01-08 22:56:06.458659+08 |           0 |        15525 |    33.48 |        48 |          0
|          -1 |               0 |              -1 |      110169581 |              -1 |      17648 |          23 |
2206 |           5 |        123 |           0
userg1              | 2020-01-08 22:56:06.460252+08 |           0 |        13972 |    33.48 |        48 |          0
|          -1 |               0 |              -1 |      136106277 |              -1 |      17648 |          23 |
2206 |           5 |        123 |           0
```

For the system catalog in **GS_WLM_USER_RESOURCE_HISTORY**, data in the **PG_TOTAL_USER_RESOURCE_INFO** view is periodically saved to historical tables only when the GUC parameter **enable_user_metric_persistent** is enabled.

For details about each column, see **GS_WLM_USER_RESOURCE_HISTORY**.

12.2 Resource Pool Monitoring

Overview

In the multi-tenant management framework, if queries are associated with resource pools, the resources occupied by the queries are summarized to the associated resource pools. You can query the real-time resource usage of all resource pools in the resource pool monitoring view and query the historical resource usage of resource pools in the resource pool monitoring history table.

The resource pool monitoring data is updated every 5s. However, due to the time difference between CNs and DNs, the actual monitoring data update time may be longer than 5s. Generally, the time does not exceed 10s. The resource pool monitoring data is persisted every 30 seconds. The resource pool monitoring logic is basically the same as that of the user resource monitoring. Therefore, the **enable_user_metric_persistent** and **user_metric_retention_time** parameters are used to control the persistence and aging of resource pool monitoring data, respectively.

Resources monitored by a resource pool include the running and queuing information of fast and slow lane jobs, and CPU, memory, and logical I/O resource monitoring information. The monitoring views and history tables are as follows:

1. Real-time monitoring view of resource pools (single CN): **GS_RESPOOL_RUNTIME_INFO**
2. Real-time monitoring view of resource pools (all CNs): **PGXC_RESPOOL_RUNTIME_INFO**
3. Real-time monitoring view of resource pool resources (single CN): **GS_RESPOOL_RESOURCE_INFO**
4. Real-time monitoring view of resource pool resources (all CNs): **PGXC_RESPOOL_RESOURCE_INFO**
5. Historical resource monitoring table of the resource pool (single CN):
6. Monitoring view of historical resource pool resources (all CNs):

NOTE

- Resource pool monitoring monitors the CPU, I/O, and memory usage of all jobs on the fast and slow lanes.
- Currently, the memory and CPU usage of fast track jobs are not controlled. When the fast lane jobs occupy a large number of resources, the used resources may exceed the resource limit.
- In the monitoring view of DN resource pools, I/O, memory, and CPU display the resource usage and limits of resource pools.
- In the monitoring view of CN resource pools, I/O, memory, and CPU display the total resource usage and limit of all DN resource pools in the cluster.
- Resource pool monitoring information on DNs is updated every 5 seconds. CNs collect resource pool monitoring information from DNs every 5 seconds. Because each instance updates or collects resource pool monitoring information independently, the monitoring information update time on each instance may be different.
- The auxiliary thread automatically invokes the persistence function every 30 seconds to make the resource pool monitoring data persistent. So, normally, you don't need to do this.

Procedure

- Querying the real-time running status of jobs in a resource pool.
  ```
  SELECT * FROM GS_RESPOOL_RUNTIME_INFO;
  ```
  The result view is as follows:
  ```
   nodegroup |    rpname    | ref_count | fast_run | fast_wait | slow_run | slow_wait
  -----------+--------------+-----------+----------+-----------+----------+-----------
   vc1       | p2           |        10 |        0 |         0 |        0 |         0
   vc2       | p3           |        10 |        5 |         5 |        0 |         0
   vc2       | p4           |         0 |        0 |         0 |        0 |         0
   vc1       | default_pool |         0 |        0 |         0 |        0 |         0
   vc2       | default_pool |         0 |        0 |         0 |        0 |         0
   vc1       | p1           |        20 |        5 |         5 |        3 |         7
  (6 rows)
  ```
  Where,
  a. **ref_count** indicates the number of jobs that reference the current resource pool information. Its value will be retained until the management ends.
  b. **fast_run** and **slow_run** are load management accounting information. Their values are valid only when **fast_limit** and **slow_limit** are larger than **0**.
  c. This view is valid only on CNs. The persistence information is stored in **GS_RESPOOL_RESOURCE_HISTORY**.
  d. For details about each field, see **GS_RESPOOL_RUNTIME_INFO**.
- Querying the resource quota and real-time resource usage of a resource pool.
  ```
  SELECT * FROM GS_RESPOOL_RESOURCE_INFO;
  ```
  The result view is as follows:
  ```
  nodegroup |    rpname    |       cgroup        | ref_count | fast_run | fast_wait | fast_limit | slow_run | 
  slow_wait | slow_limit | used_cpu | cpu_limit | used_mem | estimate_mem | mem_limit |read_kbytes | 
  write_kbytes | read_counts | write_counts | read_speed | write_speed
  -----------+--------------+---------------------+-----------+----------+-----------+------------+----------
  +-----------+------------+----------+-----------+----------+--------------+-----------+-------------+--------------
  +-------------+--------------+------------+-------------
   vc1       | p2           | DefaultClass:Rush   |        10 |        0 |         0 |         -1 |        0 |         0 |         10
  |     9.97 |         48 |       20 |            0 |     11555 |           8 |        2880 |           1 |         360 |           1
  |         589
   vc2       | p3           | DefaultClass:Rush   |        10 |        5 |         5 |          5 |        0 |         0 |         10
  |     4.98 |         48 |       11 |            0 |     11555 |           0 |         848 |           0 |         106 |           0
  |         173
   vc2       | p4           | DefaultClass:Rush   |         0 |        0 |         0 |         -1 |        0 |         0 |         10
  |        0 |         48 |        0 |            0 |     11555 |           0 |           0 |           0 |           0 |           0
  |           0
   vc1       | default_pool | DefaultClass:Medium |         0 |        0 |         0 |         -1 |        0 |         0
  |        -1 |          0 |        48 |        0 |            0 |     11555 |           0 |           0 |           0 |           0
  |           0 |           0
   vc2       | default_pool | DefaultClass:Medium |         0 |        0 |         0 |         -1 |        0 |         0
  |        -1 |          0 |        48 |        0 |            0 |     11555 |           0 |           0 |           0 |           0
  |           0 |           0
   vc1       | p1           | DefaultClass:Rush   |        20 |        5 |         5 |          5 |        3 |         7 |          3
  |     7.98 |         48 |       16 |          768 |     11555 |           8 |        2656 |           1 |         332 |           1
  |         543
  (6 rows)
  ```
  a. This view is valid on both CNs and DNs. The CPU, memory, and I/O usage on a DN indicates the resource consumption of the DN. The CPU, memory, and I/O usage on a CN is the total resource consumption of all DNs in the cluster.
  b. **estimate_mem** is valid only on CNs under dynamic load management. It displays the estimated memory accounting of the resource pool.

c. I/O monitoring information is recorded only when **enable_logical_io_statistics** is enabled.

d. For details about each field, see **GS_RESPOOL_RESOURCE_INFO**.

- Querying the resource quota and historical resource usage of a resource pool.

```
SELECT * FROM GS_RESPOOL_RESOURCE_HISTORY ORDER BY timestamp DESC;
```

The result view is as follows:

```
timestamp          |  nodegroup   |    rpname    |       cgroup        | ref_count | fast_run | fast_wait |
fast_limit | slow_run | slow_wait | slow_limit | used_cpu | cpu_limit | used_mem | estimate_mem |
mem_limit | read_kbytes | write_kbytes | read_counts | write_counts | read_speed | write_speed
------------------------------+-------------+-------------+--------------------+----------+----------
+-----------+------------+----------+-----------+------------+----------+-----------+----------+-------------
+-----------+-------------+--------------+-------------+--------------+------------+------------
 2022-03-04 09:41:57.53739+08  | vc1         | p2          | DefaultClass:Rush   |        10 |        0 |         0
|         -1 |        0 |         0 |         10 |     9.97 |        48 |   20 |            0 |     11555 |          0 |
2320 |           0 |          290 |          0 |         474
 2022-03-04 09:41:57.53739+08  | vc1         | p1          | DefaultClass:Rush   |        20 |        5 |         5
|          5 |        3 |         7 |          3 |     7.98 |        48 |   16 |          768 |     11555 |          0 |
1896 |           0 |          237 |          0 |         387
 2022-03-04 09:41:57.53739+08  | vc2         | default_pool | DefaultClass:Medium |         0 |        0
|         0 |        -1 |         0 |         0 |         -1 |         0 |        48 |    0 |            0 |     11555 |          0
|           0 |           0 |           0 |          0 |          0
 2022-03-04 09:41:57.53739+08  | vc1         | default_pool | DefaultClass:Medium |         0 |        0
|         0 |        -1 |         0 |         0 |         -1 |         0 |        48 |    0 |            0 |     11555 |          0
|           0 |           0 |           0 |          0 |          0
 2022-03-04 09:41:57.53739+08  | vc2         | p4          | DefaultClass:Rush   |         0 |        0 |         0
|         -1 |        0 |         0 |         10 |        0 |        48 |    0 |            0 |     11555 |          0 |          0
|          0 |           0 |          0 |          0
 2022-03-04 09:41:57.53739+08  | vc2         | p3          | DefaultClass:Rush   |        10 |        5 |         5
|          5 |        0 |         0 |         10 |     4.99 |        48 |   11 |            0 |     11555 |          0 |        880
|          0 |         110 |          0 |         180
 2022-03-04 09:41:27.335234+08 | vc2         | p3          | DefaultClass:Rush   |        10 |        5 |         5
|          5 |        0 |         0 |         10 |     4.98 |        48 |   11 |            0 |     11555 |          0 |        856
|          0 |         107 |          0 |         175
```

a. The monitoring information comes from the resource pool monitoring history table. When **enable_user_metric_persistent** is enabled, the monitoring information is recorded every 30 seconds.

b. The storage duration of the table data is specified by the **user_metric_retention_time** parameter.

c. For details about each field, see **GS_RESPOOL_RESOURCE_HISTORY**.

12.3 Monitoring Memory Resources

Monitoring the Memory

GaussDB(DWS) provides a view for monitoring the memory usage of the entire cluster.

Query the pgxc_total_memory_detail view as a user with sysadmin permissions.

```
SELECT * FROM pgxc_total_memory_detail;
```

If the following error message is returned during the query, enable the memory management function.

```
SELECT * FROM pgxc_total_memory_detail;
ERROR:  unsupported view for memory protection feature is disabled.
CONTEXT:  PL/pgSQL function pgxc_total_memory_detail() line 12 at FOR over EXECUTE statement
```

You can set **enable_memory_limit** and **max_process_memory** on the GaussDB(DWS) console to enable memory management. The procedure is as follows:

1. Log in to the GaussDB(DWS) management console.
2. In the navigation pane on the left, click **Clusters**.
3. In the cluster list, find the target cluster and click its name. The **Basic Information** page is displayed.
4. Click the **Parameter Modification** tab, change the value of **enable_memory_limit** to **on**, and click **Save** to save the file.
5. Change the value of **max_process_memory** to a proper one. For details about the modification suggestions, see **max_process_memory**. After it is done, click **Save**.
6. In the **Modification Preview** dialog box, confirm the modifications and click **Save**. After the modification, restart the cluster for the modification to take effect.

Monitoring the Shared Memory

You can query the context information about the shared memory on the pg_shared_memory_detail view.

```
SELECT * FROM pg_shared_memory_detail;
          contextname           | level |             parent             | totalsize | freesize | usedsize
--------------------------------+-------+--------------------------------+-----------+----------+----------
 ProcessMemory                  |     0 |                                |     24576 |     9840 |    14736
 Workload manager memory context |     1 | ProcessMemory                  |   2105400 |     7304 |  2098096
 wlm collector hash table       |     2 | Workload manager memory context |      8192 |     3736 |     4456
 Resource pool hash table       |     2 | Workload manager memory context |     24576 |    15968 |     8608
 wlm cgroup hash table          |     2 | Workload manager memory context |     24576 |    15968 |     8608
(5 rows)
```

This view lists the context name of the memory, level, the upper-layer memory context, and the total size of the shared memory.

In the database, GUC parameter **memory_tracking_mode** is used to configure the memory statistics collecting mode, including the following options:

- **none:** The memory statistics collecting function is not enabled.
- **normal:** Only memory statistics is collected in real time and no file is generated.
- **executor:** The statistics file is generated, containing the context information about all allocated memory used on the execution layer.

  When the parameter is set to **executor**, cvs files are generated under the **pg_log** directory of the DN process. The file names are in the format of **memory_track_***<DN name>***_query_***<queryid>***.csv**. The information about the operators executed by the postgres thread of the executor and all stream threads are input in this file during task execution.

  The instance is built with a file content similar to the following:

```
0, 0, ExecutorState, 0, PortalHeapMemory, 0, 40K, 602K, 23
1, 3, CStoreScan_29360131_25, 0, ExecutorState, 1, 265K, 554K, 23
2, 128, cstore scan per scan memory context, 1, CStoreScan_29360131_25, 2, 24K, 24K, 23
3, 127, cstore scan memory context, 1, CStoreScan_29360131_25, 2, 264K, 264K, 23
4, 7, InitPartitionMapTmpMemoryContext, 1, CStoreScan_29360131_25, 2, 31K, 31K, 23
5, 2, VecPartIterator_29360131_24, 0, ExecutorState, 1, 16K, 16K, 23
0, 0, ExecutorState, 0, PortalHeapMemory, 0, 24K, 1163K, 20
1, 3, CStoreScan_29360131_22, 0, ExecutorState, 1, 390K, 1122K, 20
2, 20, cstore scan per scan memory context, 1, CStoreScan_29360131_22, 2, 476K, 476K, 20
3, 19, cstore scan memory context, 1, CStoreScan_29360131_22, 2, 264K, 264K, 20
4, 7, InitPartitionMapTmpMemoryContext, 1, CStoreScan_29360131_22, 2, 23K, 23K, 20
5, 2, VecPartIterator_29360131_21, 0, ExecutorState, 1, 16K, 16K, 20
```

The fields include the output SN, SN of the memory allocation context within the thread, name of the current memory context, output SN of the parent memory context, name of the parent memory context, tree layer No. of the memory context, peak memory used by the current memory context, peak memory used by the current memory context and all its child memory contexts, and plan node ID of the query where the thread is executed.

In this example, the record "1, 3, CStoreScan_29360131_22, 0, ExecutorState, 1, 390K, 1122K, 20" represents the following information about Explain Analyze:

– **CstoreScan_29360131_22** indicates the CstoreScan operator.
– **1122K** indicates the peak memory used by the CstoreScan operator.

- **fullexec:** The generated file includes the information about all memory contexts requested by the execution layer.

  If the parameter is set to **fullexec**, the output information will be similar to that for **executor**, except that some memory context allocation information may be returned because the information about all memory applications (no matter succeeded or not) is printed. As only the memory application information is recorded, the peak memory used by the memory context is recorded as **0**.

12.4 Instance Resource Monitoring

GaussDB(DWS) provides system catalogs for monitoring the resource usage of CNs and DNs (including memory, CPU usage, disk I/O, process physical I/O, and process logical I/O), and system catalogs for monitoring the resource usage of the entire cluster.

For details about the system catalog **GS_WLM_INSTANCE_HISTORY**, see GS_WLM_INSTANCE_HISTORY.

NOTE

Data in the system catalog**GS_WLM_INSTANCE_HISTORY** is distributed in corresponding instances. CN monitoring data is stored in the CN instance, and DN monitoring data is stored in the DN instance. The DN has a standby node. When the primary DN is abnormal, the monitoring data of the DN can be restored from the standby node. However, a CN has no standby node. When a CN is abnormal and then restored, the monitoring data of the CN will be lost.

Procedure

- Query the latest resource usage of the current instance.

```
SELECT * FROM GS_WLM_INSTANCE_HISTORY ORDER BY TIMESTAMP DESC;
```

  The query result is as follows:

```
instancename |           timestamp           | used_cpu | free_mem | used_mem | io_await | io_util  |
disk_read | disk_write | process_read | process_write | logical_read | logical_write | read_counts |
write_counts
--------------+-------------------------------+----------+----------+----------+----------+----------+-----------
+------------+--------------+---------------+--------------+---------------+-------------+--------------
dn_6015_6016 | 2022-01-10 17:29:17.329495+08 |        0 |    14570 |     8982 |  662.923 |  99.9601 |
697666 |    93655.5 |       183104 |         30082 |       285659 |         30079 |      357717 |        37667
dn_6015_6016 | 2022-01-10 17:29:07.312049+08 |        0 |    14578 |     8974 |  883.102 |  99.9801 |
756228 |    81417.4 |       189722 |         30786 |       285681 |         30780 |      358103 |        38584
dn_6015_6016 | 2022-01-10 17:28:57.284472+08 |        0 |    14583 |     8969 |  727.135 |  99.9801 |
648581 |    88799.6 |       177120 |         31176 |       252161 |         31175 |      316085 |        39079
```

```
dn_6015_6016 | 2022-01-10 17:28:47.256613+08 |        0 |    14591 |     8961 |  679.534 |   100.08 |
655360 |    169962 |       179404 |        30424 |       242002 |         30422 |      303351 |        38136
```

- Query the resource usage of the current instance during a specified period.

```
SELECT * FROM GS_WLM_INSTANCE_HISTORY WHERE TIMESTAMP > '2022-01-10' AND TIMESTAMP
< '2020-01-11' ORDER BY TIMESTAMP DESC;
```

  The query result is as follows:

```
instancename |           timestamp           | used_cpu | free_mem | used_mem | io_await | io_util  |
disk_read | disk_write | process_read | process_write | logical_read | logical_write | read_counts |
write_counts
--------------+-------------------------------+----------+----------+----------+----------+----------+-----------
+------------+--------------+---------------+--------------+---------------+-------------+--------------
dn_6015_6016 | 2022-01-10 17:29:17.329495+08 |        0 |    14570 |     8982 |  662.923 |  99.9601 |
697666 |    93655.5 |       183104 |        30082 |       285659 |         30079 |      357717 |        37667
dn_6015_6016 | 2022-01-10 17:29:07.312049+08 |        0 |    14578 |     8974 |  883.102 |  99.9801 |
756228 |    81417.4 |       189722 |        30786 |       285681 |         30780 |      358103 |        38584
dn_6015_6016 | 2022-01-10 17:28:57.284472+08 |        0 |    14583 |     8969 |  727.135 |  99.9801 |
648581 |    88799.6 |       177120 |        31176 |       252161 |         31175 |      316085 |        39079
dn_6015_6016 | 2022-01-10 17:28:47.256613+08 |        0 |    14591 |     8961 |  679.534 |   100.08 |
655360 |    169962 |       179404 |        30424 |       242002 |         30422 |      303351 |        38136
```

- To query the latest resource usage of a cluster, you can invoke the **pgxc_get_wlm_current_instance_info** stored procedure on the CN.

```
SELECT * FROM pgxc_get_wlm_current_instance_info('ALL');
```

  The query result is as follows:

```
instancename |           timestamp           | used_cpu | free_mem | used_mem | io_await | io_util |
disk_read | disk_write | process_read | process_write | logical_read | logical_write | read_counts |
write_counts
--------------+-------------------------------+----------+----------+----------+----------+---------+-----------
+------------+--------------+---------------+--------------+---------------+-------------+--------------
coordinator2 | 2020-01-14 21:58:29.290894+08 |        0 |    12010 |      278 |  16.0445 | 7.19561 |
184.431 |    27959.3 |            0 |            10 |            0 |             0 |           0 |            0
coordinator3 | 2020-01-14 21:58:27.567655+08 |        0 |    12000 |      288 |  .964557 | 3.40659 |
332.468 |    3375.02 |           26 |            13 |            0 |             0 |           0 |            0
datanode1    | 2020-01-14 21:58:23.900321+08 |        0 |    11899 |      389 |  1.17296 |    3.25 |
329.6 |     2870.4 |           28 |             8 |           13 |             3 |          18 |            6
datanode2    | 2020-01-14 21:58:32.832989+08 |        0 |    11904 |      384 |   17.948 | 8.52148 |
214.186 |    25894.1 |           28 |            10 |           13 |             3 |          18 |            6
datanode3    | 2020-01-14 21:58:24.826694+08 |        0 |    11894 |      394 |  1.16088 |    3.15 |       328
|     2868.8 |           25 |            10 |           13 |             3 |          18 |            6
coordinator1 | 2020-01-14 21:58:33.367649+08 |        0 |    11988 |      300 |  9.53286 |   10.05 |
43.2 |      55232 |            0 |             0 |            0 |             0 |           0 |            0
coordinator1 | 2020-01-14 21:58:23.216645+08 |        0 |    11988 |      300 |  1.17085 | 3.21182 |
324.729 |    2831.13 |            8 |            13 |            0 |             0 |           0 |            0
(7 rows)
```

- To query historical resource usage of a cluster, you can invoke the **pgxc_get_wlm_current_instance_info** stored procedure on the CN.

```
SELECT * FROM pgxc_get_wlm_history_instance_info('ALL', '2020-01-14 21:00:00', '2020-01-14
22:00:00', 3);
```

  The query result is as follows:

```
instancename |           timestamp           | used_cpu | free_mem | used_mem | io_await |  io_util  |
disk_read | disk_write | process_read | process_write | logical_read | logical_write | read_counts |
write_counts
--------------+-------------------------------+----------+----------+----------+----------+-----------+-----------
+------------+--------------+---------------+--------------+---------------+-------------+--------------
coordinator2 | 2020-01-14 21:50:49.778902+08 |        0 |    12020 |      268 |  .127371 |   .789211 |
15.984 |    3994.41 |            0 |             0 |            0 |             0 |           0 |            0
coordinator2 | 2020-01-14 21:53:49.043646+08 |        0 |    12018 |      270 |  30.2902 |   8.65404 |
276.77 |    16741.8 |            3 |             1 |            0 |             0 |           0 |            0
coordinator2 | 2020-01-14 21:57:09.202654+08 |        0 |    12018 |      270 |   .16051 |   .979021 |
59.9401 |       5596 |            0 |             0 |            0 |             0 |           0 |            0
coordinator3 | 2020-01-14 21:38:48.948646+08 |        0 |    12012 |      276 | .0769231 | .00999001
|          0 |    35.1648 |            0 |             1 |            0 |             0 |           0 |            0
coordinator3 | 2020-01-14 21:40:29.061178+08 |        0 |    12012 |      276 |  .118421 |  .0199601
|          0 |    970.858 |            0 |             0 |            0 |             0 |           0 |            0
```

```
coordinator3 | 2020-01-14 21:50:19.612777+08 |        0 |    12010 |      278 |   24.411 |   11.7665 |
8.78244 |    44641.1 |            0 |            0 |            0 |            0 |          0 |           0
datanode1    | 2020-01-14 21:49:42.758649+08 |        0 |    11909 |      379 |  .798776 |      8.02 |
51.2 |    20924.8 |            0 |            0 |            0 |            0 |          0 |           0
datanode1    | 2020-01-14 21:49:52.760188+08 |        0 |    11909 |      379 |  23.8972 |      14.1 |
0 |      74760 |            0 |            0 |            0 |            0 |          0 |           0
datanode1    | 2020-01-14 21:50:22.769226+08 |        0 |    11909 |      379 |  39.5868 |       7.4 |        0
|    19760.8 |            0 |            0 |            0 |            0 |          0 |           0
datanode2    | 2020-01-14 21:58:02.826185+08 |        0 |    11905 |      383 |  .351648 |       .32 |
20.8 |      504.8 |            0 |            0 |            0 |            0 |          0 |           0
datanode2    | 2020-01-14 21:56:42.80793+08  |        0 |    11906 |      382 |  .559748 |       .04 |        0
|      326.4 |            0 |            0 |            0 |            0 |          0 |           0
datanode2    | 2020-01-14 21:45:21.632407+08 |        0 |    11901 |      387 |  12.1313 |   4.55544 |
3.1968 |    45177.2 |            0 |            0 |            0 |            0 |          0 |           0
datanode3    | 2020-01-14 21:58:14.823317+08 |        0 |    11898 |      390 |  .378205 |       .99 |
48 |    23353.6 |            0 |            0 |            0 |            0 |          0 |           0
datanode3    | 2020-01-14 21:47:50.665028+08 |        0 |    11901 |      387 |  1.07494 |      1.19 |
0 |    15506.4 |            0 |            0 |            0 |            0 |          0 |           0
datanode3    | 2020-01-14 21:51:21.720117+08 |        0 |    11903 |      385 |  10.2795 |      3.11 |
0 |    11031.2 |            0 |            0 |            0 |            0 |          0 |           0
coordinator1 | 2020-01-14 21:42:59.121945+08 |        0 |    12020 |      268 | .0857143 |  .0699301
|          0 |    6579.02 |            0 |            0 |            0 |            0 |          0 |           0
coordinator1 | 2020-01-14 21:41:49.042646+08 |        0 |    12020 |      268 |  20.9039 |   11.3786 |
6042.76 |    57903.7 |            0 |            0 |            0 |            0 |          0 |           0
coordinator1 | 2020-01-14 21:41:09.007652+08 |        0 |    12020 |      268 | .0446429 |    .03996 |
0 |    1109.29 |            0 |            0 |            0 |            0 |          0 |           0
(18 rows)
```

12.5 Real-time Top SQL

You can query real-time Top SQL in real-time resource monitoring views at different levels. The real-time resource monitoring view records the resource usage (including memory, data flushed to disks, and CPU time) and performance alarm information during job running.

The following table describes the external interfaces of the real-time views.

Table 12-1 Real-time resource monitoring views

Level	Monitored Node	View
Query level/perf level	Current CN	**GS_WLM_SESSION_STATISTICS**
	All CNs	**PGXC_WLM_SESSION_STATISTICS**
Operator level	Current CN	**GS_WLM_OPERATOR_STATISTICS**
	All CNs	**PGXC_WLM_OPERATOR_STATISTICS**

NOTE

- The view level is determined by the resource monitoring level, that is, the **resource_track_level** configuration.
- The perf and operator levels affect the values of the **query_plan** and **warning** columns in **GS_WLM_SESSION_STATISTICS** or **PGXC_WLM_SESSION_INFO**. For details, see SQL Self-Diagnosis.
- Prefixes **gs** and **pgxc** indicate views showing single CN information and those showing cluster information, respectively. Common users can log in to a CN in the cluster to query only views with the **gs** prefix.
- When you query this type of views, there will be network latency, because the views obtain resource usage in real time.
- If an instance fault occurs, some Top SQL statement information may fail to be recorded in real-time resource monitoring views.
- Top SQL statements are recorded in real-time resource monitoring views as follows:
  - Special DDL statements, such as **SET**, **RESET**, **SHOW**, **ALTER SESSION SET**, and **SET CONSTRAINTS**, are not recorded.
  - DDL statements, such as **CREATE**, **ALTER**, **DROP**, **GRANT**, **REVOKE**, and **VACUUM**, are recorded.
  - DML statements are recorded, including:
    - the execution of **SELECT**, **INSERT**, **UPDATE**, and **DELETE**
    - the execution of **EXPLAIN ANALYZE** and **EXPLAIN PERFORMANCE**
    - the use of the query-level or perf-level views
  - The entry statements for invoking functions and stored procedures are recorded. When the GUC parameter **enable_track_record_subsql** is enabled, some internal statements (except the **DECLARE** definition statement) of a stored procedure can be recorded. Only the internal statements delivered to DNs for execution are recorded, and the remaining internal statements are filtered out.
  - The anonymous block statement is recorded. When the GUC parameter **enable_track_record_subsql** is enabled, some internal statements of an anonymous block can be recorded. Only the internal statements delivered to DNs for execution are recorded, and the remaining internal statements are filtered out.
  - The cursor statements are recorded. If a cursor does not read data from the cache but triggers the condition for delivering the statement to a DN for execution, the cursor statement is recorded and the statement and execution plan are enhanced. However, if the cursor reads data from the cache, the cursor statement is not recorded. When a cursor statement is used in an anonymous block or function and the cursor reads a large amount of data from a DN but is not fully used, the monitoring information about the cursor on the DN cannot be recorded due to the current architecture limitation. The **With Hold** cursor syntax has a special execution logic. It executes queries during transaction committing. If a statement execution error is reported during this period of time, the **aborted** status of the job cannot be recorded in the TopSQL history table.
  - Statistics are not collected for jobs in the redistribution process.
  - The parameters of a statement with placeholders executed by JDBC are generally specified. However, if the length of the parameter and the original statement exceeds 64 KB, the parameter is not recorded. If the statement is a lightweight statement, it is directly delivered to the DN for execution and the parameter is not recorded.
  - Scheduled task statements are not recorded. This function is supported only in versions later than 8.2.1.

Prerequisites

- The GUC parameter **enable_resource_track** is set to **on**. The default value is **on**.

- The GUC parameter **resource_track_level** is set to **query**, **perf**, or **operator**. The default value is **query**.
- Job monitoring rules are as follows:
  - Jobs whose execution cost estimated by the optimizer is greater than or equal to **resource_track_cost**.
- If the Cgroups function is properly loaded, you can run the **gs_cgroup -P** command to view information about Cgroups.
- The GUC parameter **enable_track_record_subsql** specifies whether to record internal statements of a stored procedure or anonymous block.

In the preceding prerequisites, **enable_resource_track** is a system-level parameter that specifies whether to enable resource monitoring. **resource_track_level** is a session-level parameter. You can set the resource monitoring level of a session as needed. The following table describes the values of the two parameters.

Table 12-2 Setting the resource monitoring level to collect statistics

enable_resource_track	resource_track_level	Query-Level Information	Operator-Level Information
on(default)	none	Not collected	Not collected
	query(default)	Collected	Not collected
	perf	Collected	Not collected
	operator	Collected	Collected
off	none/query/operator	Not collected	Not collected

Procedure

Step 1 Query for the real-time CPU information in the **gs_session_cpu_statistics** view.

```
SELECT * FROM gs_session_cpu_statistics;
```

Step 2 Query for the real-time memory information in the **gs_session_memory_statistics** view.

```
SELECT * FROM gs_session_memory_statistics;
```

Step 3 Query for the real-time resource information about the current CN in the **gs_wlm_session_statistics** view.

```
SELECT * FROM gs_wlm_session_statistics;
```

Step 4 Query for the real-time resource information about all CNs in the **pgxc_wlm_session_statistics** view.

```
SELECT * FROM pgxc_wlm_session_statistics;
```

Step 5 Query for the real-time resource information about job operators on the current CN in the **gs_wlm_operator_statistics** view.

```
SELECT * FROM gs_wlm_operator_statistics;
```

Step 6 Query for the real-time resource information about job operators on all CNs in the **pgxc_wlm_operator_statistics** view.

```
SELECT * FROM pgxc_wlm_operator_statistics;
```

Step 7 Query for the load management information about the jobs executed by the current user in the **PG_SESSION_WLMSTAT** view.

```
SELECT * FROM pg_session_wlmstat;
```

Step 8 Query the job execution status of the current user on each CN in the **pgxc_wlm_workload_records** view (this view is available when the dynamic load function is enabled, that is, **enable_dynamic_workload** is set to **on**).

```
SELECT * FROM pgxc_wlm_workload_records;
```

----End

12.6 Historical Top SQL

You can query historical Top SQL in historical resource monitoring views. The historical resource monitoring view records the resource usage (including memory, data spilled to disks, and CPU time), running status (including errors, termination, and exceptions), and performance alarm information when a job is complete. For queries that abnormally terminate due to FATAL or PANIC errors, their status is displayed as **aborted** and no detailed information is recorded. Status information about query parsing in the optimization phase cannot be monitored.

The following table describes the external interfaces of the historical views.

Level	Monitored Node	View	
Query level/perf level	Current CN	History (Internal dump interface. Only statements that have ended in the last three minutes are displayed.)	GS_WLM_SESSION_HISTORY
		History (all statements)	GS_WLM_SESSION_INFO
	All CNs	History (Internal dump interface. Only statements that have ended in the last three minutes are displayed.)	PGXC_WLM_SESSION_HISTORY
		History (all statements)	PGXC_WLM_SESSION_INFO
Operator level	Current CN	History (Only statements that have ended in the last three minutes are displayed.)	GS_WLM_OPERATOR_HISTORY
		History (internal dump interface, all statements)	GS_WLM_OPERAROR_INFO

Level	Monitored Node	View	
	All CNs	History (Only statements that have ended in the last three minutes are displayed.)	**PGXC_WLM_OPERATOR_HISTORY**
		History (internal dump interface, all statements)	**PGXC_WLM_OPERATOR_INFO**

NOTE

- The view level is determined by the resource monitoring level, that is, the **resource_track_level** configuration.
- The perf and operator levels affect the values of the **query_plan** and **warning** columns in **GS_WLM_SESSION_STATISTICS**/**PGXC_WLM_SESSION_INFO**. For details, see SQL Self-Diagnosis.
- Prefixes **gs** and **pgxc** indicate views showing single CN information and those showing cluster information, respectively. Common users can log in to a CN in the cluster to query only views with the **gs** prefix.
- If instance fault occurs, some SQL statement information may fail to be recorded in historical resource monitoring views.
- In some abnormal cases, the status information column in the historical Top SQL may be displayed as **unknown**. The recorded monitoring information may be inaccurate.
- The SQL statements that can be recorded in historical resource monitoring views are the same as those recorded in real-time resource monitoring views. For details, see **SQL statements recorded in real-time resource monitoring views**.
- Historical Top SQL records data only when the GUC parameter **enable_resource_record** is enabled.
- You can query historical Top SQL queries and operator-level data only through the PostgreSQL database.
- Historical Top SQL focuses on locating and demarcating query performance problems. It is not used for auditing or recording syntax analysis error statements.

Prerequisites

- The GUC parameter **enable_resource_track** is set to **on**. The default value is **on**.
- The GUC parameter **resource_track_level** is set to **query**, **perf**, or **operator**. The default value is **query**. For details, see **Table 12-2**.
- The GUC parameter **enable_resource_record** is set to **on**. The default value is **on**.
- The value of the **resource_track_duration** parameter (**60s** by default) is less than the job execution time.
- The GUC parameter **enable_track_record_subsql** specifies whether to record internal statements of a stored procedure or anonymous block. The default value is **off**.
- Jobs whose execution time recorded in the real-time resource monitoring view (see **Table 12-1**) is greater than or equal to **resource_track_duration** are monitored.

- If the Cgroups function is properly loaded, you can run the **gs_cgroup -P** command to view information about Cgroups.

Procedure

Step 1 Query the load records of the current CN after its latest job is complete in the **gs_wlm_session_history** view.

```
SELECT * FROM gs_wlm_session_history;
```

Step 2 Query the load records of all the CNs after their latest job are complete in the **pgxc_wlm_session_history** view.

```
SELECT * FROM pgxc_wlm_session_history;
```

Step 3 Query the load records of the current CN through the **gs_wlm_session_info** table after the task is complete. To query the historical records successfully, set **enable_resource_record** to **on**.

```
SELECT * FROM gs_wlm_session_info;
```

- Top 10 queries that consume the most memory (You can specify a query period.)

```
SELECT * FROM gs_wlm_session_info order by max_peak_memory desc limit 10;
SELECT * FROM gs_wlm_session_info WHERE start_time >= '2022-05-15 21:00:00' and finish_time
<='2022-05-15 23:30:00' order by max_peak_memory desc limit 10;
```

- Showing the 10 queries consuming the most CPU resources:

```
SELECT * FROM gs_wlm_session_info order by total_cpu_time desc limit 10;
SELECT * FROM gs_wlm_session_info WHERE start_time >= '2022-05-15 21:00:00' and finish_time
<='2022-05-15 23:30:00' order by total_cpu_time desc limit 10;
```

Step 4 Query for the load records of all the CNs after their jobs are complete in the **pgxc_wlm_session_info** view. To query the historical records successfully, set **enable_resource_record** to **on**.

```
SELECT * FROM pgxc_wlm_session_info;
```

- Query the top 10 queries that take up the most CN processing time (You can specify a query period.)

```
SELECT * FROM pgxc_wlm_session_info order by duration desc limit 10;
SELECT * FROM pgxc_wlm_session_info WHERE start_time >= '2022-05-15 21:00:00' and finish_time
<='2022-05-15 23:30:00' order by nodename,max_peak_memory desc limit 10;
```

- Queries the execution information about a query statement that has been executed. For example, query the execution information about the statement whose **queryid** is **76561193695026478**.

```
SELECT * FROM pgxc_wlm_session_info where queryid = '76561193695026478';
```

Step 5 Use the **pgxc_get_wlm_session_info_bytime** function to filter and query the **pgxc_wlm_session_info** view. To query the historical records successfully, set **enable_resource_record** to **on**. You are advised to use this function if the view contains a large number of records.

NOTE

A GaussDB(DWS) cluster uses the UTC time by default, which has an 8-hour time difference with the system time. Before queries, ensure that the database time is the same as the system time.

- Return the queries started between **2019-09-10 15:30:00** and **2019-09-10 15:35:00** on all CNs. For each CN, a maximum of 10 queries will be returned.

```
SELECT * FROM pgxc_get_wlm_session_info_bytime('start_time', '2019-09-10 15:30:00', '2019-09-10
15:35:00', 10);
```

- Return the queries ended between **2019-09-10 15:30:00** and **2019-09-10 15:35:00** on all CNs. For each CN, a maximum of 10 queries will be returned.

```
SELECT * FROM pgxc_get_wlm_session_info_bytime('finish_time', '2019-09-10 15:30:00', '2019-09-10
15:35:00', 10);
```

Step 6 Query the recent resource information of the job operators on the current CN in the **gs_wlm_operator_history** view. Ensure that **resource_track_level** is set to **operator**.

```
SELECT * FROM gs_wlm_operator_history;
```

Step 7 Query the recent resource information of the job operators on all the CNs in the **pgxc_wlm_operator_history** view. Ensure that **resource_track_level** is set to **operator**.

```
SELECT * FROM pgxc_wlm_operator_history;
```

Step 8 Query the recent resource information of the job operators on the current CN in the **gs_wlm_operator_info** view. Ensure that **resource_track_level** is set to **operator** and **enable_resource_record** to **on**.

```
SELECT * FROM gs_wlm_operator_info;
```

Step 9 Query for the historical resource information of job operators on all the CNs in the **pgxc_wlm_operator_info** view. Ensure that **resource_track_level** is set to **operator** and **enable_resource_record** to **on**.

```
SELECT * FROM pgxc_wlm_operator_info;
```

----End

NOTE

- The number of data records that can be retained in the memory is limited due to the preset memory limit. After the real-time query is complete, the data records are imported to historical views. For a query-level view, when the number of queries to be recorded exceeds the upper limit allowed by the memory, the current query cannot be recorded and the next query is performed based on a new rule. On each CN, the memory usage of the query-level historical view is recorded (100 MB by default). You can query the data in the **PG_TOTAL_MEMORY_DETAIL** view.
- For operator-level views, whether a record can be stored depends on the upper limit allowed by the memory at that time point. If the number of plan nodes plus the number of records in the memory exceeds the upper limit, the record cannot be stored. On each CN, the maximum numbers of real-time and historical operator-level records that can be stored in the memory are **max_oper_realt_num** (set to **56987** by default) and **max_oper_hist_num** (set to **113975** by default), respectively. The average number of plan nodes of a query is **num_plan_node**. Maximum number of concurrent tasks allowed by real-time views on each CN is: **num_realt_active** = **max_oper_realt_num**/ **num_plan_node**. Maximum number of concurrent tasks allowed by historical views on each CN is: **num_hist_active** = **max_oper_hist_num**/(**180**/**run_time**)/**num_plan_node**.
- In high concurrency, ensure that the number of queries to be recorded does not exceed the maximum values set for query- and operator-level views. You can modify the memory of the historical query view by configuring the **session_history_memory** parameter. The memory size increases in direct proportion to the maximum number of queries that can be recorded.

12.7 TopSQL Query Example

In this section, TPC-DS sample data is used as an example to describe how to query **Real-time Top SQL** and **Historical Top SQL**.

Configuring Cluster Parameters

To query for historical or archived resource monitoring information about jobs of top SQLs, you need to set related GUC parameters first. The procedure is as follows:

1. Log in to the GaussDB(DWS) management console.
2. On the **Cluster Management** page, locate the required cluster and click the cluster name. The cluster details page is displayed.
3. Click the **Parameter Modifications** tab to view the values of cluster parameters.
4. Set an appropriate value for parameter **resource_track_duration** and click **Save**.

   NOTE

   If **enable_resource_record** is set to **on**, storage space expansion may occur and thereby slightly affects the performance. Therefore, set is to **off** if record archiving is unnecessary.
5. Go back to the **Cluster Management** page, click the refresh button in the upper right corner, and wait until the cluster parameter settings are applied.

Example for Querying for Top SQLs

The TPC-DS sample data is used as an example.

Step 1 Open the SQL client tool and connect to your database.

Step 2 Run the **EXPLAIN** statement to query for the estimated cost of the SQL statement to be executed to determine whether resources of the SQL statement will be monitored.

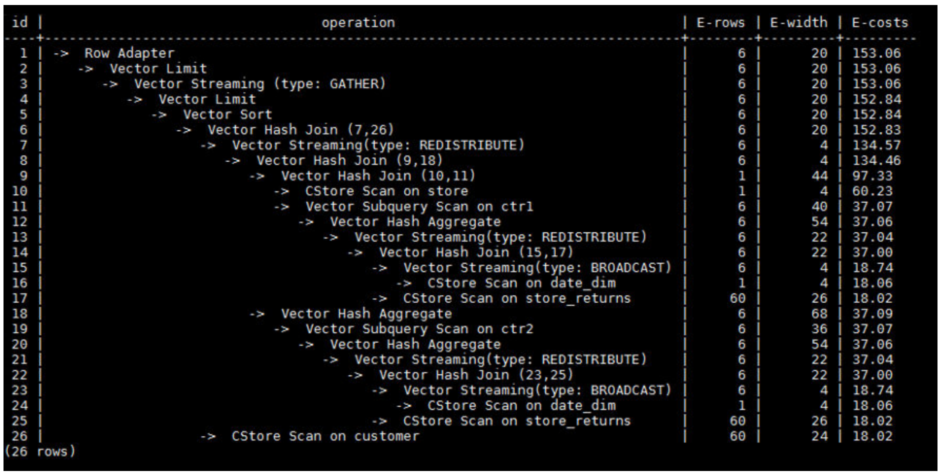
By default, only resources of a query whose execution cost is greater than the value of **resource_track_cost** are monitored and can be queried by users.

For example, run the following statements to query for the estimated execution cost of the SQL statement:

```
SET CURRENT_SCHEMA = tpcds;
EXPLAIN WITH customer_total_return AS
( SELECT sr_customer_sk as ctr_customer_sk,
sr_store_sk as ctr_store_sk,
sum(SR_FEE) as ctr_total_return
FROM store_returns, date_dim
WHERE sr_returned_date_sk = d_date_sk AND d_year =2000
GROUP BY sr_customer_sk, sr_store_sk )
SELECT  c_customer_id
FROM customer_total_return ctr1, store, customer
WHERE ctr1.ctr_total_return > (select avg(ctr_total_return)*1.2
FROM customer_total_return ctr2
WHERE ctr1.ctr_store_sk = ctr2.ctr_store_sk)
AND s_store_sk = ctr1.ctr_store_sk
AND s_state = 'TN'
AND ctr1.ctr_customer_sk = c_customer_sk
ORDER BY c_customer_id
limit 100;
```

In the following query result, the value in the first row of the **E-costs** column is the estimated cost of the SQL statement.

Figure 12-1 EXPLAIN result



```
id |                                  operation                                   | E-rows | E-width | E-costs
---+------------------------------------------------------------------------------+--------+---------+---------
 1 |  ->  Row Adapter                                                              |      6 |      20 | 153.06
 2 |     ->  Vector Limit                                                          |      6 |      20 | 153.06
 3 |        ->  Vector Streaming (type: GATHER)                                    |      6 |      20 | 153.06
 4 |           ->  Vector Limit                                                    |      6 |      20 | 152.84
 5 |              ->  Vector Sort                                                  |      6 |      20 | 152.84
 6 |                 ->  Vector Hash Join (7,26)                                   |      6 |      20 | 152.83
 7 |                    ->  Vector Streaming(type: REDISTRIBUTE)                   |      6 |       4 | 134.57
 8 |                       ->  Vector Hash Join (9,18)                             |      6 |       4 | 134.46
 9 |                          ->  Vector Hash Join (10,11)                         |      1 |      44 | 97.33
10 |                             ->  CStore Scan on store                          |      1 |       4 | 60.23
11 |                             ->  Vector Subquery Scan on ctr1                  |      6 |      40 | 37.07
12 |                                ->  Vector Hash Aggregate                      |      6 |      54 | 37.06
13 |                                   ->  Vector Streaming(type: REDISTRIBUTE)    |      6 |      22 | 37.04
14 |                                      ->  Vector Hash Join (15,17)             |      6 |      22 | 37.00
15 |                                         ->  Vector Streaming(type: BROADCAST) |      6 |       4 | 18.74
16 |                                            ->  CStore Scan on date_dim        |      1 |       4 | 18.06
17 |                                         ->  CStore Scan on store_returns      |     60 |      26 | 18.02
18 |                          ->  Vector Hash Aggregate                            |      6 |      68 | 37.09
19 |                             ->  Vector Subquery Scan on ctr2                  |      6 |      36 | 37.07
20 |                                ->  Vector Hash Aggregate                      |      6 |      54 | 37.06
21 |                                   ->  Vector Streaming(type: REDISTRIBUTE)    |      6 |      22 | 37.04
22 |                                      ->  Vector Hash Join (23,25)             |      6 |      22 | 37.00
23 |                                         ->  Vector Streaming(type: BROADCAST) |      6 |       4 | 18.74
24 |                                            ->  CStore Scan on date_dim        |      1 |       4 | 18.06
25 |                                         ->  CStore Scan on store_returns      |     60 |      26 | 18.02
26 |                    ->  CStore Scan on customer                                |     60 |      24 | 18.02
(26 rows)
```

In this example, to demonstrate the resource monitoring function of TopSQL, you need to set **resource_track_cost** to a value smaller than the estimated cost in the **EXPLAIN** result, for example, **100**. For details about the parameter setting, see **resource_track_cost**.

NOTE

After completing this example, you still need to reset **resource_track_cost** to its default value **100000** or a proper value. An overly small parameter value will compromise the database performance.

Step 3 Run SQL statements.

```
SET CURRENT_SCHEMA = tpcds;
WITH customer_total_return AS
(SELECT sr_customer_sk as ctr_customer_sk,
sr_store_sk as ctr_store_sk,
sum(SR_FEE) as ctr_total_return
FROM store_returns,date_dim
WHERE sr_returned_date_sk = d_date_sk
AND d_year =2000
GROUP BY sr_customer_sk ,sr_store_sk)
SELECT  c_customer_id
FROM customer_total_return ctr1, store, customer
WHERE ctr1.ctr_total_return > (select avg(ctr_total_return)*1.2
FROM customer_total_return ctr2
WHERE ctr1.ctr_store_sk = ctr2.ctr_store_sk)
AND s_store_sk = ctr1.ctr_store_sk
AND s_state = 'TN'
AND ctr1.ctr_customer_sk = c_customer_sk
ORDER BY c_customer_id
limit 100;
```

Step 4 During statement execution, query for the real-time memory peak information about the SQL statement on the current CN.

```
SELECT query,max_peak_memory,average_peak_memory,memory_skew_percent FROM
gs_wlm_session_statistics ORDER BY start_time DESC;
```

The preceding command queries for the real-time peak information at the query-level. The peak information includes the maximum memory peak among all DNs per second, average memory peak among all DNs per second, and memory usage skew across DNs.

For more examples of querying for the real-time resource monitoring information of top SQLs, see **Real-time Top SQL**.

Step 5 Wait until the SQL statement execution in **Step 3** is complete, and then query for the historical resource monitoring information of the statement.

```
SELECT query,start_time,finish_time,duration,status FROM gs_wlm_session_history ORDER BY start_time
DESC;
```

The preceding command queries for the historical information at the query-level. The peak information includes the execution start time, execution duration (unit: ms), and execution status. The time unit is ms.

For more examples of querying for the historical resource monitoring information of top SQLs, see **Historical Top SQL**.

Step 6 Wait for 3 minutes after the execution of the SQL statement in **Step 3** is complete, query for the historical resource monitoring information of the statement in the **info** view.

If **enable_resource_record** is set to **on** and the execution time of the SQL statement in **Step 3** is no less than the value of **resource_track_duration**, historical information about the SQL statement will be archived to the **gs_wlm_session_info** view 3 minutes after the execution of the SQL statement is complete.

The **info** view can be queried only when the **postgres** database is connected. Therefore, switch to the **postgres** database before running the following statement:

```
SELECT query,start_time,finish_time,duration,status FROM gs_wlm_session_info ORDER BY start_time desc;
```

----End

13 GaussDB(DWS) Performance Tuning

13.1 Overview

Database performance tuning is the process of optimizing database system configuration and SQL queries to improve database performance and efficiency. The purpose includes eliminating performance bottlenecks, reducing response times, increasing throughput and resource utilization, cutting costs, and improving system stability.

This section provides comprehensive guidance for DBAs on performance diagnosis, system tuning, and SQL tuning, as well as practical examples of SQL tuning.

Precautions

- Database performance tuning is a complex and intricate process. To achieve the optimal performance and efficiency, performance tuning must take into consideration multiple factors, such as hardware, software, queries, configuration, and data structures. Engineers performing the performance tuning must be familiar with how database systems work in great detail, including a deep understanding of the system software architecture, software and hardware configurations, database configuration parameters, concurrency control, query handling, and database applications.
- Performance tuning sometimes requires a cluster restart, which may interrupt services. To avoid that, you are advised to schedule performance tuning tasks that require a cluster restart to occur during off-peak hours.

Performance Tuning Process

Figure 13-1 illustrates the performance tuning process.

Figure 13-1 GaussDB(DWS) performance tuning

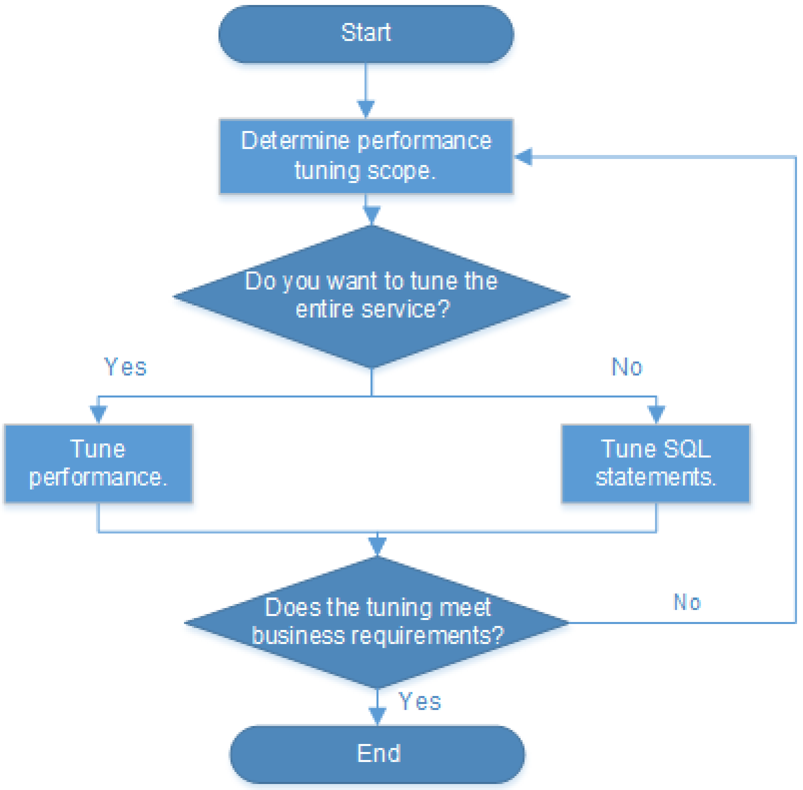

Start
Determine performance tuning scope.
Do you want to tune the entire service?
Yes
No
Tune performance.
Tune SQL statements.
Does the tuning meet business requirements?
No
Yes
End

Table 13-1 gives a brief introduction to each phase of the performance tuning process.

Table 13-1 Phase-by-phase introduction to GaussDB(DWS) performance tuning

Phase	Description
Performance Diagnosis	Obtain the CPU, memory, I/O, and network resource usage of each node to check whether these resources are fully utilized and whether any performance bottlenecks exist.
System Optimization	Perform OS and database system-level performance tuning to achieve better utilization of existing CPU, memory, I/O, and network resources, prevent resource conflicts, and improve query throughput.

Phase	Description
SQL Tuning	Analyze the SQL statements used and 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: Identify possible causes by analyzing the execution plan and perform specific optimization by modifying database-level SQL optimization parameters. ● Compiling better SQL statements: Compile better SQL statements in the scenarios, such as cache of intermediate and temporary data for complex queries, result set cache, and result set combination.

13.2 Performance Diagnosis

13.2.1 Cluster Performance Analysis

The node specifications of different GaussDB(DWS) clusters may vary in terms of the number of CPU cores, memory capacity, and node storage capacity. Different specifications lead to different service handling capacity and performance. Before creating a cluster, you need to select the appropriate cluster specifications based on the actual workloads and application scenario.

If the workloads increase, more resources (such as CPU, memory, and network bandwidth) will be needed in order to maintain the same level of database performance. Insufficient cluster resources will lead to performance issues.

GaussDB(DWS) provides abundant monitoring metrics that you can use to monitor cluster performance and status, including CPU usage, memory usage, disk usage, disk I/O, and network I/O. For any abnormality, you can check the metrics to locate the root cause.

If your service requires additional compute or storage resources, expand the capacity of an existing cluster by adding more nodes to it or changing node specifications through the management console.

13.2.2 Slow SQL Analysis

13.2.2.1 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 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;
```

After the query, query statements are returned as a list, ranked by execution time in descending order. The first result is the query statement that has the longest execution time in the system. The returned result contains the SQL statement invoked by the system and the SQL statement 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 2 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 3 View the running query statements.

Viewing **pg_stat_activity** 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 4 Analyze the status of the query statements that were run for a long time.

- If the query statement is normal, wait until the execution 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 command output lists a query statement in the block state. The lock resource requested by this query statement is occupied by another session, so this query statement is waiting for the session to release the lock resource.

NOTE

Only when the query is blocked by internal lock resources, the **waiting** field is **true**. In most cases, block happens when query statements are waiting for lock resources to be released. However, query statements may be blocked because they are waiting to write in files or for timers. Such blocked queries are not displayed in the **pg_stat_activity** view.

----End

13.2.2.2 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 View blocked query statements and information 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 information, query status, as well as information about the tables and schemas that block the query statements are returned.

Step 2 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 a command output 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

13.2.3 SQL Diagnosis

GaussDB(DWS) clusters support SQL diagnosis, which shows the complete execution plans of specific SQL queries. You can search for SQL queries (such as slow queries) using a combination of multiple filter criteria.

To use SQL diagnosis, perform the following steps:

Step 1 Log in to the GaussDB(DWS) console.

Step 2 Choose **Dedicated Clusters** > **Clusters** and locate the cluster to be monitored.

Step 3 In the **Operation** column of the target cluster, click **Monitoring Panel**.

Step 4 In the navigation pane on the left, choose **Utilities** > **SQL Diagnosis**. The metrics include:

- Query ID
- Database
- Schema Name
- User Name
- Client
- Client IP Address
- Running Time (ms)
- CPU Time (ms)
- Scale-Out Started
- Completed
- Details

Step 5 On the **SQL Diagnosis** page, you can view the SQL diagnosis information. In the **Details** column of a specified query ID, click **View** to view the detailed SQL diagnosis result, including:

- Alarm Information
- SQL Statement
- Execution Plan

----End

13.2.4 Table Diagnosis

GaussDB(DWS) provides statistics and diagnostic tools for you to learn table status, including:

- Skew Rate: monitors and analyzes uneven data distribution in a cluster, and displays information about the 50 largest tables whose skew rate is higher than 5%.
- Dirty Page Rate: monitors and analyzes dirty pages in a cluster, and displays information about the 50 largest tables whose dirty page rate is higher than 50%.

Skew Rate

Improper distribution columns can cause severe skew during operator computing or data spill to disk. The workloads will be unevenly distributed on DNs, resulting in high disk usage on individual DNs and affecting their performance. After identifying tables with a high skew rate and a relatively large size, you can reselect distribution columns for these tables to have their data redistributed. For details, see **How Do I Change Distribution Columns?**

Procedure

Step 1 Log in to the GaussDB(DWS) console.

Step 2 Choose **Dedicated Clusters** > **Clusters** and locate the cluster to be monitored.

Step 3 In the **Operation** column of the target cluster, click **Monitoring Panel**.

Step 4 In the navigation tree on the left, choose **Utilities** > **Table Diagnosis** and click the **Skew Rate** tab. The tables that meet the statistics collection conditions in the cluster are displayed.

----End

Dirty Page Rate

DML operations on tables may generate dirty data, which unnecessarily occupies cluster storage. You can identify tables with a high dirty page rate and a relatively large size, and handle them accordingly. For more information, see **Solution to High Disk Usage and Cluster Read-Only**.

Procedure

Step 1 Log in to the GaussDB(DWS) console.

Step 2 Choose **Dedicated Clusters** > **Clusters** and locate the cluster to be monitored.

Step 3 In the **Operation** column of the target cluster, click **Monitoring Panel**.

Step 4 In the navigation tree on the left, choose **Utilities** > **Table Diagnosis** and click the **Dirty Page Rate** tab. The tables that meet the statistics collection conditions in the cluster are displayed.

----End

13.3 System Optimization

13.3.1 Tuning Database Parameters

To ensure high performance of the database, you are advised to configure GUC parameters based on available resources and the actual workloads. This section describes some of the common parameters and the recommended configurations for them. For more details, see **Configuring GUC Parameters**.

Parameters Related to Database Memory

Table 13-2 Parameters related to database memory

GUC Parameter	Description	Configuration Suggestion
max_process_memory	Specifies the maximum physical memory available to a single CN/DN.	● On DNs, the value of this parameter is determined based on the server's physical memory and the number of DNs deployed on a single node. Parameter value = (Physical memory – **vm.min_free_kbytes**) x 0.8/(n + Number of primary DNs). This parameter aims to ensure system reliability, preventing node OOM caused by increasing memory usage. **vm.min_free_kbytes** indicates OS memory reserved for kernels to receive and send data. Its value is at least 5% of the total memory. That is, **max_process_memory** = Physical memory x 0.8/ (n + Number of primary DNs). If the cluster scale (number of nodes in the cluster) is smaller than 256, n=1; if the cluster scale is larger than 256 and smaller than 512, n=2; if the cluster scale is larger than 512, n=3. ● Set this parameter on CNs to the same value as that on DNs. ● RAM is the maximum memory allocated to the cluster.

GUC Parameter	Description	Configuration Suggestion
shared_buffers	Specifies the size of the shared memory used by GaussDB(DWS). If the value of this parameter is increased, GaussDB(DWS) requires more System V shared memory than the default system setting.	It is recommended that **shared_buffers** be set 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. Set this parameter to a large value for row storage and a small value for column storage. For column-store tables: shared_buffers = (Memory of a single server/Number of DNs on the single server) x 0.4 x 0.25 If you want to increase the value of **shared_buffers**, you also need to increase the value of **checkpoint_segments**, because a longer period of time is required to write a large amount of new or changed data.
cstore_buffers	Specifies the size of the shared buffer used by column-store tables and column-store tables (ORC, Parquet, and CarbonData) of OBS and HDFS foreign tables.	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**. Use **cstore_buffers** to specify the cache of ORC, Parquet, or CarbonData metadata and data for OBS or HDFS foreign tables. The metadata cache size should be 1/4 of **cstore_buffers** and not exceed 2 GB. The remaining cache is shared by column-store data and foreign table column-store data.

GUC Parameter	Description	Configuration Suggestion
work_mem	Specifies the size of the memory used by internal sequential operations and the Hash table before data is written into temporary disk files.	The default value is 512 MB for small-scale memory (**max_process_memory** is less than 30 GB) and 2 GB for large-scale memory (**max_process_memory** is greater than or equal to 30 GB). When the specified physical memory is insufficient, **work_mem** determines whether to write additional operator calculation data into temporary tables based on query characteristics and concurrency. This reduces performance by five to ten times and increases query response times from seconds to minutes. ● In complex serial query scenarios, each query requires five to ten associated operations. Set **work_mem** using the following formula: **work_mem** = 50% of the memory/10. ● In simple serial query scenarios, 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, use the formula: **work_mem** = **work_mem** in serialized scenario/Number of concurrent SQL statements.

GUC Parameter	Description	Configuration Suggestion
maintenance_work_mem	Specifies the maximum size of memory used for maintenance operations, involving **VACUUM**, **CREATE INDEX**, and **ALTER TABLE ADD FOREIGN KEY**.	If you set this parameter to the value of **work_mem**, database dump files can be cleaned up and restored more efficiently. In a database session, only one maintenance operation can be performed at a time. Maintenance is usually performed when there are not much sessions. When the automatic cleanup process is running, up to **autovacuum_max_workers** times of the memory will be allocated. In this case, set **maintenance_work_mem** to a value greater than or equal to that of **work_mem**.

Parameters Related to Queue Concurrency in Databases

GUC Parameter	Description	Configuration Suggestion
max_active_statements (global concurrent queue)	Controls the maximum number of concurrent jobs on a single CN.	All common users' jobs are subject to this threshold, regardless of their complexity. When the number of concurrent jobs reaches the specified threshold, the excess jobs have to wait in a queue. Administrator's jobs are exempt from this limit. Set the value of this parameter based on system resources, such as CPU, I/O, and memory resources, to ensure that the system resources can be fully utilized and the system will not be crashed due to excessive concurrent jobs.
parctl_min_cost (local concurrent queue)	Controls the maximum number of concurrent jobs within the same resource pool on a single CN.	The number of concurrent complex jobs are controlled based on their cost.

NOTE

When tuning the **max_active_statements** parameter (global concurrent queue), pay attention to the following:

- If **max_active_statements** is set to **-1**, which indicates that global concurrency is not limited, users may be disconnected in a high concurrency scenario.
- In a point query scenario, set **max_active_statements** to **100**.
- In an analytical query scenario, set **max_active_statements** to the number of CPU cores divided by the number of DNs. Generally, its value ranges from 5 to 8.

Database Communication Parameters

By default, nodes in a database cluster communicate using the TCP proxy communication library.

Table 13-3 Database communication parameters

GUC Parameter	Description	Configuration Suggestion
comm_quota_size	**comm_quota_size** controls the size of data transmitted every time in each flow channel. Its default value is **1M**.	In a high concurrency scenario, you can increase its value to improve communication performance, but doing so consumes more memory. Optimize this parameter as needed. If you query the **pg_total_memory_detail** view of a DN and find that the memory used by the communication layer has reached the threshold of **comm_usable_memory**, set **comm_quota_size** to a small value, such as **512K**.
comm_usable_memory	**comm_usable_memory** controls the memory on a DN that can be used for database communication.	The value of this parameter is only used for memory flow control. The default flow control value is 1 MB. If the memory usage exceeds half of the parameter value, the flow control value will be automatically changed to 0.5 MB. If only 20% of the memory specified by the parameter is available, the flow control value will be changed to the allowed minimum, 8 KB.

Database Connection Parameters

Table 13-4 Database connection parameters

GUC Parameter	Description	Configuration Suggestion
max_connections	Specifies the maximum number of concurrent connections to the database. This parameter affects the concurrent processing capability of the cluster.	Retain the default value of this parameter on CNs. Set this parameter on DNs to a value calculated using this formula: Number of CNs x Value of this parameter on a CN. If the value of this parameter is increased, GaussDB(DWS) may require more System V shared memory or semaphore, which may exceed the default maximum value of the OS. In this case, modify the value as needed.
max_prepared_transactions	Specifies the maximum number of transactions that can stay in the **prepared** state simultaneously. If the value of this parameter is increased, GaussDB(DWS) requires more System V shared memory than the default system setting.	The value of **max_connections** is related to **max_prepared_transactions**. Before configuring **max_connections**, ensure that the value of **max_prepared_transactions** is greater than or equal to that of **max_connections**. In this way, each session has a prepared transaction in the waiting state.
session_timeout	Specifies the maximum duration a database connection can stay idle before it is automatically disconnected.	The value can be an integer in the range 0 to 86400. The minimum unit is second (s). The value **0** disables this timeout mechanism. Generally, you are advised not to set this parameter to **0**.

Other Performance-related Parameters

Table 13-5 Other performance-related parameters

GUC Parameter	Description	Configuration Suggestion
enable_dynamic_workload	Specifies whether to enable dynamic load management. Dynamic load management refers to the automatic queue control of complex queries based on user loads in a database. This fine-tunes system parameters without manual adjustment.	This parameter is enabled by default. Notes: ● Simple query jobs (which are estimated to require less than 32 MB memory) and non-DML statements (statements other than **INSERT**, **UPDATE**, **DELETE**, and **SELECT**) have no adaptive load restrictions. Control the upper memory limits for them on a single CN using **max_active_statements**. ● In adaptive load scenarios, the value cannot be increased. If you increase it, memory cannot be controlled for certain statements, such as statements that have not been analyzed. ● Reduce concurrency in the following scenarios, because high concurrency may lead to uncontrollable memory usage. – A single tuple occupies excessive memory, for example, a base table contains a column more than 1 MB wide. – A query is fully pushed down. – A statement occupies a large amount of memory on the CN, for example, a statement that cannot be pushed down or a cursor withholding statement. – An execution plan creates a hash table based on the hash join operator, and the table has many duplicate values and occupies a large amount of memory. – UDFs are used, which occupy a large amount of memory. ● When configuring this parameter, you can set **query_dop** to **0** (adaptive). In this case, the system dynamically selects the optimal degree of parallelism (DOP) for

GUC Parameter	Description	Configuration Suggestion
		each query based on resource usage and the execution plan. The **enable_dynamic_workload** parameter supports the dynamic memory allocation.
bulk_write_ring_size	Specifies the size of a ring buffer used for parallel data import.	This parameter affects the database import performance. You are advised to increase the value of this parameter on DNs when a large amount of data is to be imported. The default value is **2GB**.
data_replicate_buffer_size	Specifies the memory used by queues when the sender sends data pages to the receiver.	The value of this parameter affects the buffer size for data replication between the primary and standby servers. The default value is 16 MB for a CN and 128 MB for a DN. If the server memory is 256 GB, you can increase the value to 512 MB.

13.3.2 SMP Parallel Execution

Complex queries may take a long time. In a system with low concurrency support, this can be a problem. SMP is used to implement operator-level parallel execution, which can effectively speed up queries, improving query performance and resource utilization.

The SMP feature improves performance through operator parallelism but may drive more resource usage, including CPU, memory, network, and I/O. In essence, SMP is a method that trades resources for time, meaning it accelerates queries at the cost of additional resources. It improves system performance in appropriate scenarios and when resources are sufficient, but may also deteriorate performance if used inappropriately. Furthermore, compared with serial processing, SMP generates more candidate plans, which is more time-consuming and may hurt performance.

The SMP feature of GaussDB(DWS) is controlled by the GUC parameter **query_dop**. Users use this parameter to specify an appropriate degree of query parallelism.

Application Scenarios and Constraints for SMP

Applicable Scenarios

- Operators supporting parallel processing are used.

  The execution plan contains the following operators:

a. Scan: Row Storage common table and a line memory partition table sequential scanning, column-oriented storage ordinary table and column-oriented storage partition table sequential scanning, HDFS internal and external table sequence scanning. Surface scanning GDS data can be imported at the same time. All of the above does not support replication tables.
b. Join: HashJoin, NestLoop
c. Agg: HashAgg, SortAgg, PlainAgg, and WindowAgg, which supports only **partition by**, and does not support **order by**.
d. Stream: Redistribute, Broadcast
e. Other: Result, Subqueryscan, Unique, Material, Setop, Append, VectoRow, RowToVec

- SMP-unique operators are used.

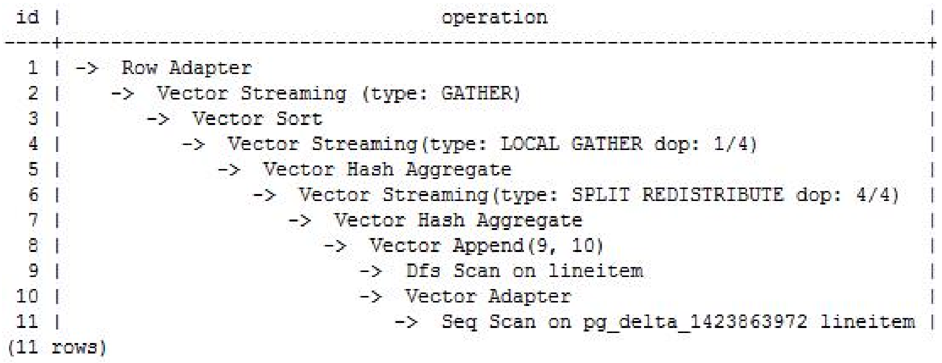
  To execute queries in parallel, Stream operators are added for data exchange for the SMP feature. These new operators can be considered as the subtypes of Stream operators.

  a. Local Gather aggregates data of parallel threads within a DN
  b. Local Redistribute redistributes data based on the distributed key across threads within a DN
  c. Local Broadcast broadcasts data to each thread within a DN.
  d. Local RoundRobin distributes data in polling mode across threads within a DN.
  e. Split Redistribute redistributes data across parallel threads on different DNs.
  f. Split Broadcast broadcasts data to all parallel DN threads in the cluster.

  Among these operators, Local operators exchange data between parallel threads within a DN, and non-Local operators exchange data across DNs.

- Example

  The TPCH Q1 parallel plan is used as an example.



```
 id |                              operation                                |
----+-----------------------------------------------------------------------+
  1 | ->  Row Adapter                                                       |
  2 |     ->  Vector Streaming (type: GATHER)                               |
  3 |         ->  Vector Sort                                               |
  4 |             ->  Vector Streaming(type: LOCAL GATHER dop: 1/4)         |
  5 |                 ->  Vector Hash Aggregate                             |
  6 |                     ->  Vector Streaming(type: SPLIT REDISTRIBUTE dop: 4/4)  |
  7 |                         ->  Vector Hash Aggregate                     |
  8 |                             ->  Vector Append(9, 10)                  |
  9 |                                 ->  Dfs Scan on lineitem              |
 10 |                                 ->  Vector Adapter                    |
 11 |                                     ->  Seq Scan on pg_delta_1423863972 lineitem |
(11 rows)
```

  In this plan, implement the Hdfs Scan and HashAgg operator parallel, and adds the Local Gather and Split Redistribute data exchange operator.

  In this example, the sixth operator is Split Redistribute, and **dop: 4/4** next to the operator indicates that the degree of parallelism of the sender and receiver is 4. 4 No operator is Local Gather, marked dop: 1/4 above, this operator sender thread parallel degree is 4, while the receiving end thread parallelism degree to 1, that is, lower-layer 5 number Hash Aggregate operators according to the 4 parallel degree, while the working mode of the port on the upper-layer 1 to 3 number operator according to the executed one by one, 4 number operator is used to achieve intra-DN concurrent threads data aggregation.

You can view the parallelism situation of each operator in the dop information.

Non-Applicable Scenarios

1. Small queries are performed, where plan generation may account for a significant portion of the total query time.
2. Operators are processed on CNs.
3. Statements that cannot be pushed down are executed.
4. The **subplan** of a query and operators containing a subquery are executed.

Impact of Resource Availability on SMP Performance

The SMP architecture accelerates queries at the cost of additional resources. After the plan parallelism is executed, more resources are consumed, including the CPU, memory, I/O, and network bandwidth. As the DOP grows, the resource consumption also increases. If these resources become a bottleneck, SMP cannot improve performance. On the contrary, it may do exactly the opposite. Adaptive SMP is provided to dynamically select the optimal parallel degree for each query based on the resource usage and query requirements. The following information describes the situations that the SMP affects theses resources:

- **CPU resources**

  In a general customer scenario, the system CPU usage rate is not high. Using the SMP parallelism architecture will fully use the CPU resource to improve the system performance. If the number of CPU kernels of the database server is too small and the CPU usage is already high, enabling the SMP parallelism may deteriorate the system performance due to resource compete between multiple threads.

- **Memory resources**

  The query parallel causes memory usage growth, but the memory upper limit used by each operator is still restricted by **work_mem**. Assume that **work_mem** is 4 GB, and the degree of parallelism is 2, then the memory upper limit of each concurrent thread is 2 GB. When **work_mem** is small or the system memory is sufficient, running SMP parallelism may push data down to disks. As a result, the query performance deteriorates.

- **Network bandwidth resources**

  To execute queries in parallel, data exchange operators are added. Local stream operators exchange data between threads within a DN. Data is exchanged in memory, so it does not impact network performance. Non-local operators exchange data over the network and increase the network load. If the capacity of a network resource has already become a bottleneck, parallelism may hurt performance.

- **I/O resources**

  A parallel scan increases I/O resource consumption. It can improve performance only when I/O resources are sufficient.

Other Factors Impacting SMP Performance

Besides the resource factor, other factors may also impact SMP performance, such as uneven data distribution across tables and the degree of system parallelism.

- **Impact of data skew on SMP performance**

  Serious data skew deteriorates parallel execution performance. For example, if the data volume of a value in the join column is much more than that of other values, the data volume of a parallel thread will be much more than that of others after Hash-based data redistribution, resulting in the long-tail issue and poor parallelism performance.

- **Impact of system parallelism degree on SMP performance**

  The SMP feature uses more resources, and unused resources are decreasing in a high concurrency scenario. Therefore, enabling the SMP parallelism will result in serious resource compete among queries. Once resource competes occur, no matter the CPU, I/O, memory, or network resources, all of them will result in entire performance deterioration. In the high concurrency scenario, enabling the SMP will not improve the performance effect and even may cause performance deterioration.

Suggestions for SMP Parameter Settings

To enable the SMP adaptation function, set **query_dop** to **0** and adjust the following parameters to obtain an optimal DOP selection:

- comm_usable_memory

  If the system memory is large, the value of **max_process_memory** is large. In this case, you are advised to set the value of this parameter to 5% of **max_process_memory**, that is, 4 GB by default.

- comm_max_stream

  The recommended value for this parameter is calculated as follows: comm_max_stream = Min(dop_limit x dop_limit x 20 x 2, max_process_memory (bytes) x 0.025/Number of DNs/260). The value must be within the value range of **comm_max_stream**.

- max_connections

  The recommended value for this parameter is calculated as follows: max_connections = dop_limit x 20 x 6 + 24. The value must be within the value range of **max_connections**.

CAUTION

In the preceding formulas, **dop_limit** indicates the number of CPUs corresponding to each DN in the cluster. It is calculated as follows: **dop_limit** = Number of logical CPU cores of a single server/Number of DNs of a single server.

SMP Configuration Procedure

NOTICE

The CPU, memory, I/O, and network bandwidth resources are sufficient. In essence, SMP is a method that trades resources for time. After the plan parallelism is executed, resource consumption increases. When these resources become a bottleneck, SMP may deteriorate, rather than improve performance. In addition, it takes a longer time to generate SMP plans than serial plans. Therefore, in TP services that mainly involve short queries or in case resources are insufficient, you are advised to disable SMP by setting **query_dop** to **1**.

Procedure:

1. Observe the current system load situation. If the resource is sufficient (the resource usage ratio is smaller than 50%), perform step 2. Otherwise, exit this system.
2. Set **query_dop** to **1** (default value). Use **explain** to generate an execution plan and check whether the plan can be used in scenarios described in **Application Scenarios and Constraints for SMP**. If the plan can be used, go to the next step.
3. Set **query_dop=-***value*. The value range of the parallelism degree is [1, *value*].
4. Set **query_dop=***value*. The parallelism degree is 1 or *value*.
5. Before the query statement is executed, set **query_dop** to an appropriate value. After the statement is executed, set **query_dop** to **off**. For example:

```
SET query_dop = 0;
SELECT COUNT(*) FROM t1 GROUP BY a;
......
SET query_dop = 1;
```

NOTE

- If resources are sufficient, the higher the degree of parallelism, the better the performance improvement result.
- The SMP parallelism degree supports a session level setting and you are advised to enable SMP before executing queries that meet the requirements. After the execution is complete, disable SMP. Otherwise, SMP may affect services during peak hours.
- SMP adaptability (**query_dop** ≤ 0) depends on resource management. If resource management is disabled (**use_workload_manager** is turned **off**), only plans with parallelism degree of only 1 or 2 will be generated.

13.3.3 Configuring LLVM

LLVM dynamic compilation can be used to generate customized machine code for each query to replace original common functions. The query performance is improved by reducing redundant judgment condition and virtual function invocation, and make local data more accurate during actual queries.

LLVM needs to consume extra time to pre-generate intermediate representation (IR) and compile it into code. Therefore, if the data volume is small or if a query itself consumes little time, LLVM actually does more harm than good.

LLVM Application Scenarios and Constraints

Applicable Scenarios

- Expressions supporting LLVM. The query statements that contain the following expressions support LLVM optimization:
  a. CASE...WHEN...
  b. IN
  c. Bool (AND/OR/NOT)
  d. BooleanTest (IS_NOT_KNOWN/IS_UNKNOWN/IS_TRUE/IS_NOT_TRUE/IS_FALSE/IS_NOT_FALSE)
  e. NullTest (IS_NOT_NULL/IS_NULL)
  f. Operators
  g. Functions (lpad, substring, btrim, rtrim, and length)
  h. Nullif

  The following data types are supported for expression calculation: bool, tinyint, smallint, int, bigint, float4, float8, numeric, date, time, timetz, timestamp, timestamptz, interval, bpchar, varchar, text, and oid.

  Consider using LLVM dynamic compilation and optimization only when expressions are used in the following scenarios:
  - **filter** on the **Scan** node in the case of a vectorized executor.
  - **complicate hash condition**, **hash join filter**, and **hash join target** in the **Hash Join** node.
  - **filter** and **join filter** in the **Nested Loop** node.
  - **merge join filter** and **merge join target** in the **Merge Join** node.
  - **filter** in the Group node.
- Operators that can use LLVM:
  a. Join: HashJoin
  b. Agg: HashAgg
  c. Sort

  Among them:
  - HashJoin supports only Hash Inner Join, and the corresponding hash cond supports comparisons between int4, bigint, and bpchar.
  - HashAgg supports sum and avg operations of bigint and numeric data types. Group By statements support int4, bigint, bpchar, text, varchar, timestamp, and the count(*) aggregation operation.
  - Sort supports only comparisons between int4, bigint, numeric, bpchar, text, and varchar data types.

  With the exception of the operations above, LLVM dynamic compilation and optimization cannot be used. To further confirm, use the explain performance tool to check.

Non-Applicable Scenarios

- LLVM dynamic compilation and optimization are not supported on CNs.

- Tables that have small amounts of data cannot be dynamically compiled using LLVM.
- Query jobs with a non-vectorized execution path cannot be generated.

Other Factors Impacting LLVM Performance

The result of LLVM optimization depends not only on operations and computation in the database, but also on the hardware environment.

- Number of C- functions invoked by query statements

  CodeGen cannot be used in all expressions in an entire expression, that is, some expressions use CodeGen while others invoke original C codes for computation. In an entire expression, if more expressions invoke original C codes, LLVM dynamic compilation and optimization may reduce the computational performance. By setting **log_min_messages** to **DEBUG1**, you can check expressions that directly invoke C codes.
- Memory resources

  One of the key LLVM features is to ensure the locality of data, that is, data should be stored in registers whenever possible. Data loading should be reduced at the same time. Therefore, when using LLVM optimization, the value of work_mem must be set as large as required to ensure that the code is processed in the memory using LLVM. Otherwise, performance may deteriorate.
- Optimizer cost estimation

  The LLVM feature realizes a simple cost estimation model. You can determine whether to use LLVM dynamic compilation and optimization for the current node based on the sizes of tables involved in node computation. If the optimizer understates the actual number of rows involved, the expected performance gains may not be realized. An overestimation will have the same effect.

Recommended Usage of LLVM

LLVM is enabled in the database kernel by default, and users can configure it based on the analysis above. The overall suggestions are as follows:

1. Set an appropriate value for **work_mem** and set it as large as possible. If much data is flushed to disks, you are advised to disable LLVM dynamic compilation and optimization by setting **enable_codegen** to **off**.
2. Set an appropriate value for **codegen_cost_threshold** (The default value is 10,000). Ensure that LLVM dynamic compilation and optimization is not used when the data volume is small. After the value is set, if the database performance deteriorates due to the use of LLVM dynamic compilation and optimization, increase the value.
3. If a large number of C- functions are invoked, you are advised to disable LLVM dynamic compilation and optimization.
4. The constants following the **In** expression cannot exceed 10. Otherwise, LLVM compilation and optimization cannot be performed.

NOTE

If resources are sufficient, the database performance will improve as the data volume increases.

13.4 SQL Tuning

13.4.1 SQL Query Execution Process

The process from receiving SQL statements to the statement execution by the SQL engine is shown in **Figure 13-2** and **Table 13-6**. The texts in red are steps where database administrators can optimize queries.

Figure 13-2 Execution process of query-related SQL statements by the SQL engine

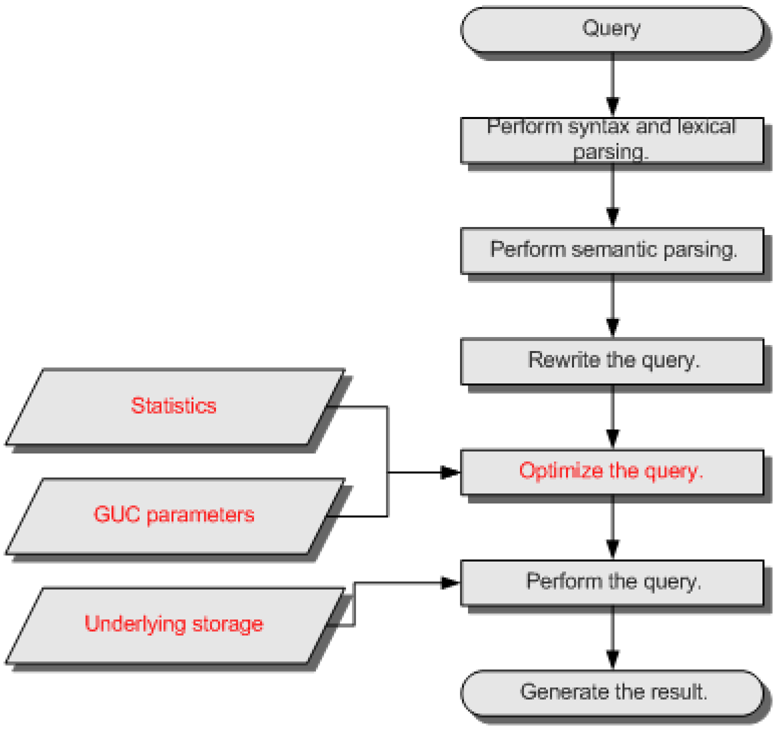


Table 13-6 Execution process of query-related SQL statements by the SQL engine

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.
3. Rewrite the query statements.	Converts the output of the last step into the structure that optimizes the query execution.

Step	Description
4. Optimize the query.	Determines the execution mode of SQL statements (the execution plan) based on the result obtained from the last step and the internal database statistics. For details about the impact of statistics and GUC parameters on 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 last 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(DWS) optimizer is a typical Cost-based Optimization (CBO). The database uses the CBO to calculate the number of tuples and execution cost for each execution step in every execution plan. This calculation is based on factors such as the number of table tuples, column width, NULL record ratio, and characteristic values (such as distinct, MCV, and HB values) using specific cost calculation methods. The database then selects the execution plan with the lowest cost for overall execution or for returning the first tuple. These characteristic values are the statistics, which is the core for optimizing a query. Accurate statistics helps the optimizer 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 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(DWS) supports nested loop, merge join, and hash join. The optimizer estimates the result set value and the execution cost under each join mode based on the statistics of the **customer** and **store_sales** tables and selects the execution plan that takes the lowest execution cost.

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(DWS) supports both row-store and column-store tables. The choice of storage mode ultimately depends on your business needs. Column-store tables are suitable for computing services that mainly involve associations and aggregations. Row-store tables are better suited for point queries and large-scale updates or deletions.

Optimization methods of each storage mode will be described in details in the performance optimization chapter.

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 practical experience.

This requires that the system administrators know the customer business well and have professional knowledge of SQL statements.

13.4.2 SQL Execution Plan

An SQL execution plan is a node tree that displays the detailed steps performed when GaussDB(DWS) executes an SQL statement.

You can run the **EXPLAIN** command to view the execution plan generated for each query by an optimizer. **EXPLAIN** outputs a row of information for each execution node, showing the basic node type and the expense estimate that the optimizer makes for executing the node.

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 common usages are as follows. For more usages, see **EXPLAIN Syntax**.

- 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 **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.

Description of common execution plan keywords:

1. Table access modes
   - Seq Scan/CStore Scan

     Scans all rows of the table in sequence. These are basic scan operators, which are used to scan row-store and column-store tables in sequence.
   - Index Scan/CStore Index Scan

     Scans indexes of row-store and column-store tables. There are indexes in row-store or column-store tables, and the condition column is the index column.

     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 first sort the location of index identifier rows based on 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

       To use a bitmap index to capture a data page, you need to scan the index to obtain the bitmap and then scan the base table.
     - 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

     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 not exceed 10,000. The table that returns a smaller subset will work as a foreign table, and indexes are recommended for connection fields 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 a merge join, data in the two joined tables is sorted by join columns. Then, data is extracted from the two tables to a sorted table for matching.

   A merge join requires more resources for sorting and its performance is lower than that of a hash join. If the source data has been sorted, it does not need to be sorted again when merge join is performed. In this case, the performance of merge join is better than that of hash join.

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 10000 rows. As a result, the cost is not decreased. It increases a bit (by 10000 x **cpu_operator_cost**) to reflect the extra CPU time spent on checking the **WHERE** condition.

   - LIMIT

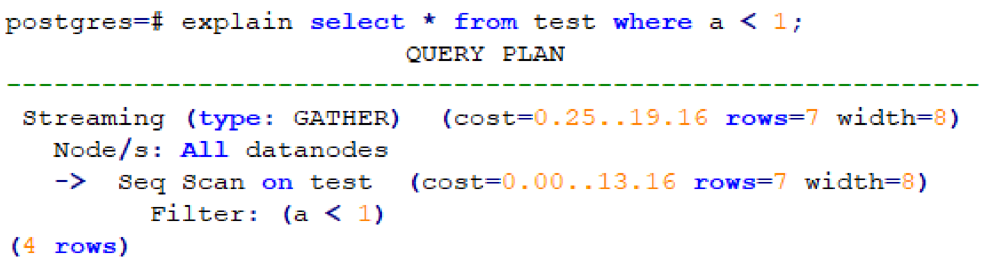
     LIMIT limits the number of output execution results. If a **LIMIT** condition is added, not all rows are retrieved.

Execution Plan Display Format

GaussDB(DWS) provides four display formats: **normal**, **pretty**, **summary**, and **run**. You can change the display format of execution plans by setting **explain_perf_mode**.

- **normal** indicates that the default printing format is used. Figure 13-3 shows the display format.

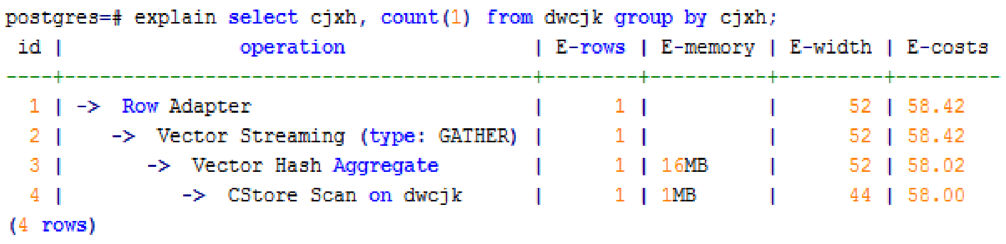
Figure 13-3 Example of an execution plan in normal format



```
postgres=# explain select * from test where a < 1;
                        QUERY PLAN
-------------------------------------------------------------
 Streaming (type: GATHER)  (cost=0.25..19.16 rows=7 width=8)
   Node/s: All datanodes
   ->  Seq Scan on test  (cost=0.00..13.16 rows=7 width=8)
         Filter: (a < 1)
(4 rows)
```

- **pretty** indicates that the optimized display mode of GaussDB(DWS) is used. A new format contains a plan node ID, directly and effectively analyzing performance. Figure 13-4 is an example.

Figure 13-4 Example of an execution plan using the pretty format


```
postgres=# explain select cjxh, count(1) from dwcjk group by cjxh;
 id |                operation                | E-rows | E-memory | E-width | E-costs
----+-----------------------------------------+--------+----------+---------+---------
  1 | ->  Row Adapter                         |      1 |          |      52 | 58.42
  2 |    ->  Vector Streaming (type: GATHER)  |      1 |          |      52 | 58.42
  3 |       ->  Vector Hash Aggregate         |      1 | 16MB     |      52 | 58.02
  4 |          ->  CStore Scan on dwcjk       |      1 | 1MB      |      44 | 58.00
(4 rows)
```

- **summary** indicates that the analysis result based on such information is printed in addition to the printed information in the format specified by **pretty**.
- **run** indicates that in addition to the printed information specified by **summary**, the database exports the information as a CSV file.

Common Types of Plans

GaussDB(DWS) has three types of distributed plans:

- Fast Query Shipping (FQS) plan

  The CN directly delivers statements to DNs. Each DN executes the statements independently and summarizes the execution results on the CN.
- Stream plan

  The CN generates a plan for the statements to be executed and delivers the plan to DNs for execution. During the execution, DNs use the Stream operator to exchange data.
- Remote-Query plan

  After generating a plan, the CN delivers some statements to DNs. Each DN executes the statements independently and sends the execution result to the CN. The CN executes the remaining statements in the plan.

The existing tables **tt01** and **tt02** are defined as follows:

```
CREATE TABLE tt01(c1 int, c2 int) DISTRIBUTE BY hash(c1);
CREATE TABLE tt02(c1 int, c2 int) DISTRIBUTE BY hash(c2);
```

Type 1: FQS plan, all statements pushed down

Two tables are joined, and the join condition is the distribution column of each table. If the stream operator is disabled, the CN directly sends statements to each DN for execution. The result is summarized on the CN.

```
SET enable_stream_operator=off;
SET explain_perf_mode=normal;

EXPLAIN (VERBOSE on,COSTS off) SELECT * FROM tt01,tt02 WHERE tt01.c1=tt02.c2;
                                    QUERY PLAN
------------------------------------------------------------------------------------------------------------
 Data Node Scan on "__REMOTE_FQS_QUERY__"
   Output: tt01.c1, tt01.c2, tt02.c1, tt02.c2
   Node/s: All datanodes
   Remote query: SELECT tt01.c1, tt01.c2, tt02.c1, tt02.c2 FROM dbadmin.tt01, dbadmin.tt02 WHERE tt01.c1
= tt02.c2
(4 rows)
```

Type 2: Non-FQS plan, some statements pushed down

Two tables are joined and the join condition contains non-distribution columns. If the stream operator is disabled, the CN delivers the base table scanning statements to each DN. Then, the JOIN operation is performed on the CN.

```
SET enable_stream_operator=off;
SET explain_perf_mode=normal;

EXPLAIN (VERBOSE on,COSTS off) SELECT * FROM tt01,tt02 WHERE tt01.c1=tt02.c1;
                              QUERY PLAN
-------------------------------------------------------------------------------
 Hash Join
   Output: tt01.c1, tt01.c2, tt02.c1, tt02.c2
   Hash Cond: (tt01.c1 = tt02.c1)
   ->  Data Node Scan on tt01 "_REMOTE_TABLE_QUERY_"
         Output: tt01.c1, tt01.c2
         Node/s: All datanodes
         Remote query: SELECT c1, c2 FROM ONLY dbadmin.tt01 WHERE true
   ->  Hash
         Output: tt02.c1, tt02.c2
         ->  Data Node Scan on tt02 "_REMOTE_TABLE_QUERY_"
               Output: tt02.c1, tt02.c2
               Node/s: All datanodes
               Remote query: SELECT c1, c2 FROM ONLY dbadmin.tt02 WHERE true
(13 rows)
```

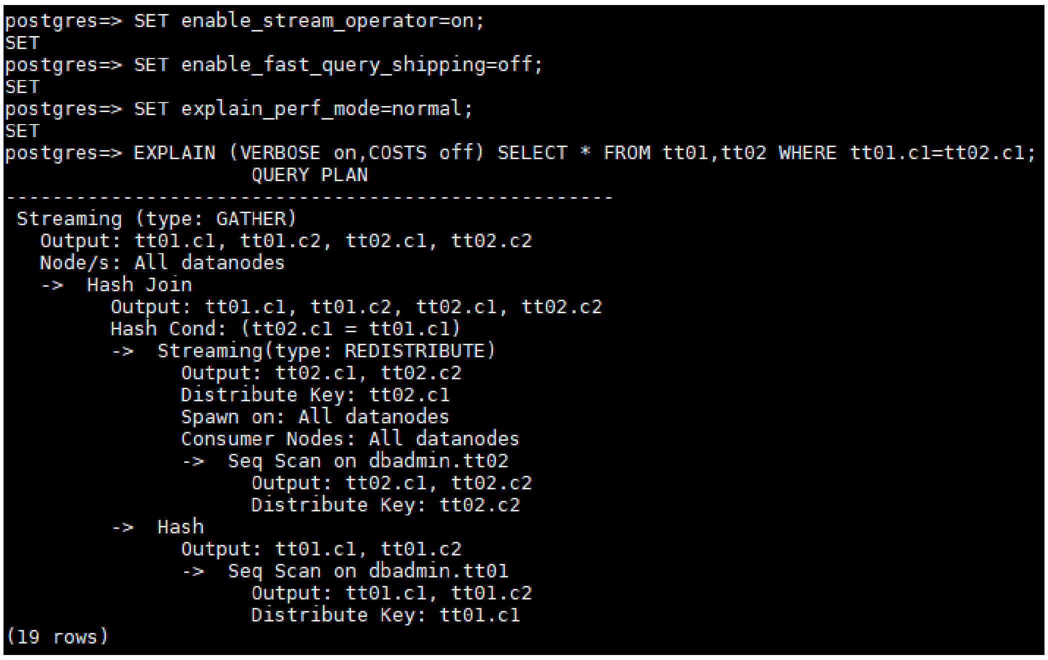
Type 3: Stream plan, no data exchange between DNs

Two tables are joined, and the join condition is the distribution column of each table. DNs do not need to exchange data. After generating a stream plan, the CN delivers the plan except the Gather Stream part to DNs for execution. The CN scans the base table on each DN, performs hash join, and sends the result to the CN.

```
SET enable_fast_query_shipping=off;
SET enable_stream_operator=on;

EXPLAIN (VERBOSE on,COSTS off) SELECT * FROM tt01,tt02 WHERE tt01.c1=tt02.c2;
                    QUERY PLAN
---------------------------------------------------
 Streaming (type: GATHER)
   Output: tt01.c1, tt01.c2, tt02.c1, tt02.c2
   Node/s: All datanodes
   ->  Hash Join
         Output: tt01.c1, tt01.c2, tt02.c1, tt02.c2
         Hash Cond: (tt01.c1 = tt02.c2)
         ->  Seq Scan on dbadmin.tt01
               Output: tt01.c1, tt01.c2
               Distribute Key: tt01.c1
         ->  Hash
               Output: tt02.c1, tt02.c2
               ->  Seq Scan on dbadmin.tt02
                     Output: tt02.c1, tt02.c2
                     Distribute Key: tt02.c2
(14 rows)
```

Type 4: Stream plan, with data exchange between DNs

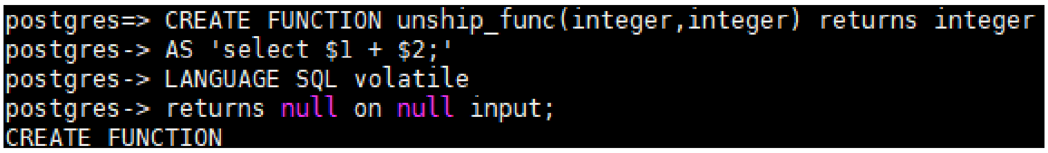
When two tables are joined and the join condition contains non-distribution columns, and the stream operator is enabled (SET enable_stream_operator=on), a stream plan is generated, which allows data exchange between DNs. For table **tt02**, the base table is scanned on each DN. After the scanning, the **Redistribute Stream** operator performs hash calculation based on **tt02.c1** in the **JOIN** condition, sends the hash calculation result to each DN, and then performs JOIN on each DN, finally, the data is summarized to the CN.



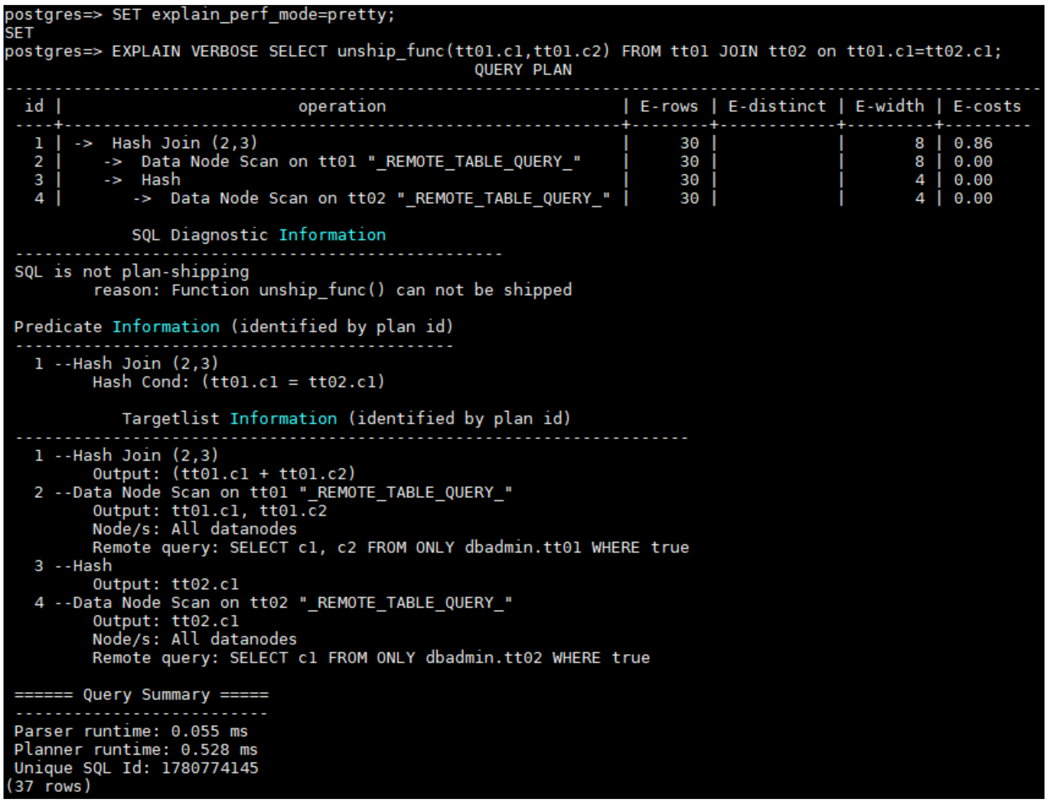
```
postgres=> SET enable_stream_operator=on;
SET
postgres=> SET enable_fast_query_shipping=off;
SET
postgres=> SET explain_perf_mode=normal;
SET
postgres=> EXPLAIN (VERBOSE on,COSTS off) SELECT * FROM tt01,tt02 WHERE tt01.c1=tt02.c1;
                     QUERY PLAN
----------------------------------------------------
 Streaming (type: GATHER)
   Output: tt01.c1, tt01.c2, tt02.c1, tt02.c2
   Node/s: All datanodes
   ->  Hash Join
         Output: tt01.c1, tt01.c2, tt02.c1, tt02.c2
         Hash Cond: (tt02.c1 = tt01.c1)
         ->  Streaming(type: REDISTRIBUTE)
               Output: tt02.c1, tt02.c2
               Distribute Key: tt02.c1
               Spawn on: All datanodes
               Consumer Nodes: All datanodes
               ->  Seq Scan on dbadmin.tt02
                     Output: tt02.c1, tt02.c2
                     Distribute Key: tt02.c2
         ->  Hash
               Output: tt01.c1, tt01.c2
               ->  Seq Scan on dbadmin.tt01
                     Output: tt01.c1, tt01.c2
                     Distribute Key: tt01.c1
(19 rows)
```

Type 5: Remote-Query plan

unship_func cannot be pushed down and does not meet partial pushdown requirements (subquery pushdown). Therefore, you can only send base table scanning statements to DNs and collect base table data to the CN for calculation.



```
postgres=> CREATE FUNCTION unship_func(integer,integer) returns integer
postgres-> AS 'select $1 + $2;'
postgres-> LANGUAGE SQL volatile
postgres-> returns null on null input;
CREATE FUNCTION
```



```
postgres=> SET explain_perf_mode=pretty;
SET
postgres=> EXPLAIN VERBOSE SELECT unship_func(tt01.c1,tt01.c2) FROM tt01 JOIN tt02 on tt01.c1=tt02.c1;
                                                QUERY PLAN
------------------------------------------------------------------------------------------------------------
 id |                      operation                      | E-rows | E-distinct | E-width | E-costs
----+-----------------------------------------------------+--------+------------+---------+---------
  1 | ->  Hash Join (2,3)                                 |     30 |            |       8 | 0.86
  2 |    ->  Data Node Scan on tt01 "_REMOTE_TABLE_QUERY_"   |     30 |            |       8 | 0.00
  3 |    ->  Hash                                         |     30 |            |       4 | 0.00
  4 |       ->  Data Node Scan on tt02 "_REMOTE_TABLE_QUERY_" |     30 |            |       4 | 0.00

             SQL Diagnostic Information
 ------------------------------------------------
 SQL is not plan-shipping
        reason: Function unship_func() can not be shipped

 Predicate Information (identified by plan id)
 ---------------------------------------------
   1 --Hash Join (2,3)
         Hash Cond: (tt01.c1 = tt02.c1)

            Targetlist Information (identified by plan id)
 --------------------------------------------------------------------
   1 --Hash Join (2,3)
         Output: (tt01.c1 + tt01.c2)
   2 --Data Node Scan on tt01 "_REMOTE_TABLE_QUERY_"
         Output: tt01.c1, tt01.c2
         Node/s: All datanodes
         Remote query: SELECT c1, c2 FROM ONLY dbadmin.tt01 WHERE true
   3 --Hash
         Output: tt02.c1
   4 --Data Node Scan on tt02 "_REMOTE_TABLE_QUERY_"
         Output: tt02.c1
         Node/s: All datanodes
         Remote query: SELECT c1 FROM ONLY dbadmin.tt02 WHERE true

 ====== Query Summary =====
 -------------------------
 Parser runtime: 0.055 ms
 Planner runtime: 0.528 ms
 Unique SQL Id: 1780774145
(37 rows)
```

EXPLAIN PERFORMANCE Description

You can use **EXPLAIN ANALYZE** or **EXPLAIN PERFORMANCE** to check the SQL statement execution information and compare the actual execution and the optimizer's estimation to find what to optimize. **EXPLAIN PERFORMANCE** provides the execution information on each DN, whereas **EXPLAIN ANALYZE** does not.

Tables are defined as follows:

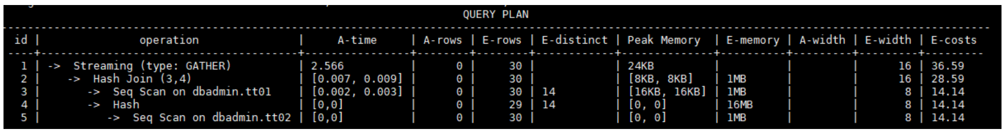
```
CREATE TABLE tt01(c1 int, c2 int) DISTRIBUTE BY hash(c1);
CREATE TABLE tt02(c1 int, c2 int) DISTRIBUTE BY hash(c2);
```

The following SQL query statement is used as an example:

```
SELECT * FROM tt01,tt02 WHERE tt01.c1=tt02.c2;
```

The output of EXPLAIN PERFORMANCE consists of the following parts:

1. Execution Plan



```
                                                             QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------
 id |                operation                |      A-time      | A-rows | E-rows | E-distinct | Peak Memory  | E-memory | A-width | E-width | E-costs
----+-----------------------------------------+------------------+--------+--------+------------+--------------+----------+---------+---------+---------
  1 | ->  Streaming (type: GATHER)            | 2.566            |      0 |     30 |            | 24KB         |          |         |      16 | 36.59
  2 |    ->  Hash Join (3,4)                  | [0.007, 0.009]   |      0 |     30 |            | [8KB, 8KB]   | 1MB      |         |      16 | 28.59
  3 |       ->  Seq Scan on dbadmin.tt01      | [0.002, 0.003]   |      0 |     30 | 14         | [16KB, 16KB] | 1MB      |         |       8 | 14.14
  4 |       ->  Hash                          | [0,0]            |      0 |     29 | 14         | [0, 0]       | 16MB     |         |       8 | 14.14
  5 |          ->  Seq Scan on dbadmin.tt02   | [0,0]            |      0 |     30 |            | [0, 0]       | 1MB      |         |       8 | 14.14
```

The plan is displayed as a table, which contains 11 columns: **id**, **operation**, **A-time**, **A-rows**, **E-rows**, **E-distinct**, **Peak Memory**, **E-memory**, **A-width**, **E-width**, and **E-costs**. Table 13-7 describes the meanings of the columns.

Table 13-7 Execution column description

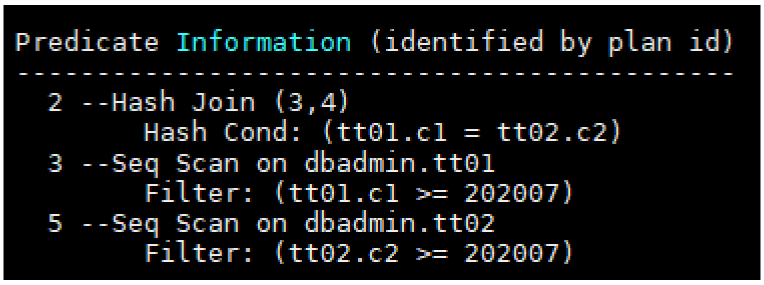
Column	Description
id	ID of an execution operator.
operation	Name of an execution operator. The operator of the Vector prefix refers to a vectorized execution engine operator, which exists in a query containing a column-store table. Streaming is a special operator. It implements the core data shuffle function of the distributed architecture. Streaming has three types, which correspond to different data shuffle functions in the distributed architecture: ● Streaming (type: GATHER): The CN collects data from DNs. ● Streaming(type: REDISTRIBUTE): Data is redistributed to all the DNs based on selected columns. ● Streaming(type: BROADCAST): Data on the current DN is broadcast to all other DNs.
A-time	Execution time of an operator on each DN. Generally, A-time of an operator is two values enclosed by square brackets ([]), indicating the shortest and longest time for completing the operator on all DNs, including the execution time of the lower-layer operators. Note: In the entire plan, the execution time of a leaf node is the execution time of the operator, while the execution time of other operators includes the execution time of its subnodes.
A-rows	Actual rows output by an operator.
E-rows	Estimated rows output by each operator.
E-distinct	Estimated distinct value of the hashjoin operator.
Peak Memory	Peak memory used when the operator is executed on each DN. The left value in [] is the minimum value, and the right value in [] is the maximum value.
E-memory	Estimated memory used by each operator on a DN. Only operators executed on DNs are displayed. In certain scenarios, the memory upper limit enclosed in parentheses will be displayed following the estimated memory usage.
A-width	The actual width of each line of tuple of the current operator. This parameter is valid only for the heavy memory operator is displayed, including: (Vec)HashJoin, (Vec)HashAgg, (Vec) HashSetOp, (Vec)Sort, and (Vec)Materialize operator. The (Vec)HashJoin calculation of width is the width of the right subtree operator, it will be displayed in the right subtree.

Column	Description
E-width	Estimated width of the output tuple of each operator.
E-costs	Estimated execution cost of each operator. ● E-costs are defined by the optimizer based on cost parameters, habitually grasping disk page as a unit. Other overhead parameters are set by referring to E-costs. ● The cost of each node (the E-costs value) includes the cost of all of its child nodes. ● Overhead reflects only what the optimizer is concerned about, but does not consider the time that the result row passed to the client. Although the time may play a very important role in the actual total time, it is ignored by the optimizer, because it cannot be changed by modifying the plan.

2. SQL Diagnostic Information

   SQL self-diagnosis information. Performance optimization points identified during optimization and execution are displayed. When **EXPLAIN** with the **VERBOSE** attribute (built-in **VERBOSE** of **EXPLAIN PERFORMANCE**) is executed on DML statements, SQL self-diagnosis information is also generated to help locate performance issues.

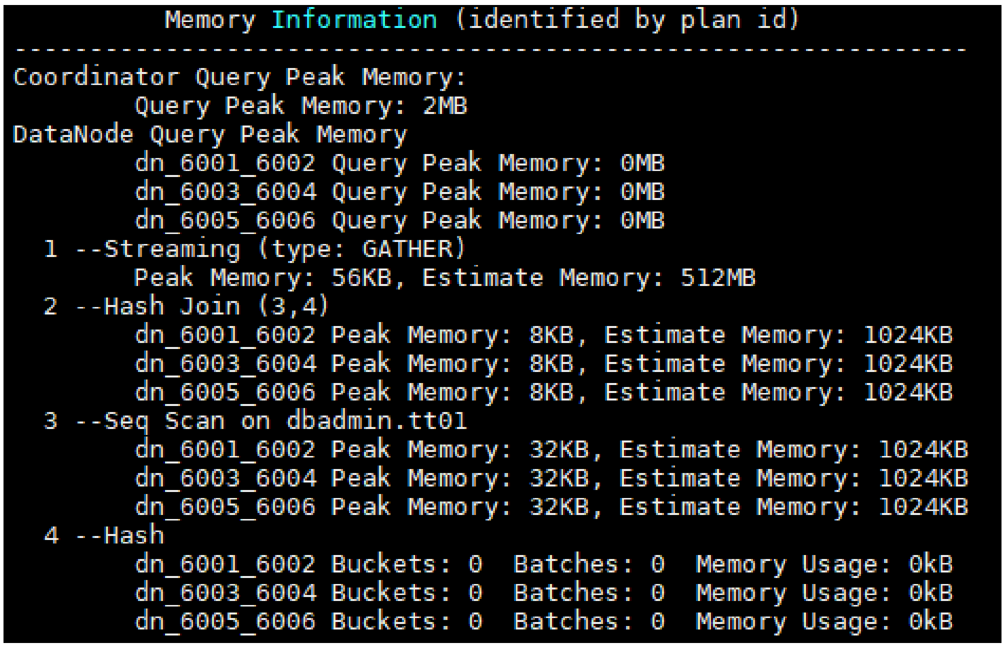
3. Predicate Information (identified by plan id)



```
Predicate Information (identified by plan id)
---------------------------------------------
  2 --Hash Join (3,4)
        Hash Cond: (tt01.c1 = tt02.c2)
  3 --Seq Scan on dbadmin.tt01
        Filter: (tt01.c1 >= 202007)
  5 --Seq Scan on dbadmin.tt02
        Filter: (tt02.c2 >= 202007)
```

   This part displays the filtering conditions of the corresponding execution operator node, that is, the information that does not change during the entire plan execution, mainly the join conditions and filter information.

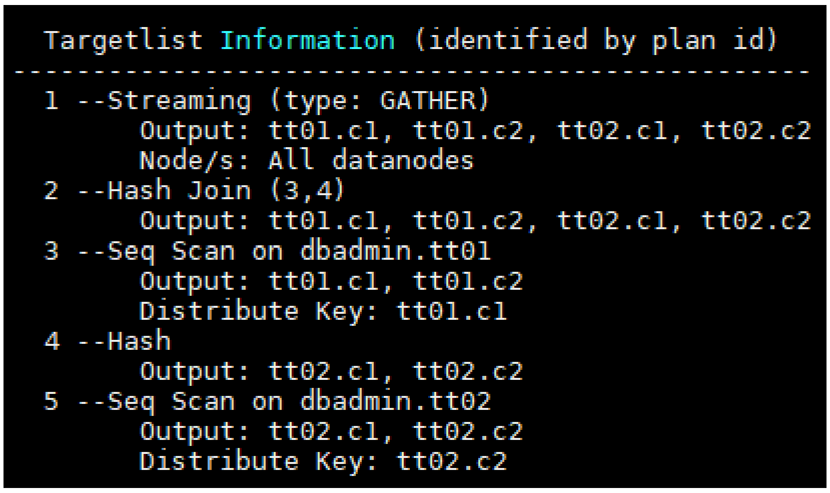
4. Memory Information (identified by plan id)



```
          Memory Information (identified by plan id)
-------------------------------------------------------------
Coordinator Query Peak Memory:
        Query Peak Memory: 2MB
DataNode Query Peak Memory
        dn_6001_6002 Query Peak Memory: 0MB
        dn_6003_6004 Query Peak Memory: 0MB
        dn_6005_6006 Query Peak Memory: 0MB
  1 --Streaming (type: GATHER)
        Peak Memory: 56KB, Estimate Memory: 512MB
  2 --Hash Join (3,4)
        dn_6001_6002 Peak Memory: 8KB, Estimate Memory: 1024KB
        dn_6003_6004 Peak Memory: 8KB, Estimate Memory: 1024KB
        dn_6005_6006 Peak Memory: 8KB, Estimate Memory: 1024KB
  3 --Seq Scan on dbadmin.tt01
        dn_6001_6002 Peak Memory: 32KB, Estimate Memory: 1024KB
        dn_6003_6004 Peak Memory: 32KB, Estimate Memory: 1024KB
        dn_6005_6006 Peak Memory: 32KB, Estimate Memory: 1024KB
  4 --Hash
        dn_6001_6002 Buckets: 0  Batches: 0  Memory Usage: 0kB
        dn_6003_6004 Buckets: 0  Batches: 0  Memory Usage: 0kB
        dn_6005_6006 Buckets: 0  Batches: 0  Memory Usage: 0kB
```

Memory Usage displays the memory usage of operators in the entire plan, mainly Hash and Sort operators, including the peak memory of operators (Peak Memory), memory estimated by the optimizer (Estimate Memory), and control memory (Control Memory), estimated memory usage (operator memory), actual width during execution (Width), number of automatic memory expansion times (Auto Spread Num), whether to spill data to disks in advance (Early Spilled), and spill information which includes the number of repeated data spills (Spill Time(s)), number of internal and foreign table partitions spilled to disks (inner/outer partition spill num), number of files spilled to disks (temp file num), amount of data spilled to disks, and amount of data flushed to the minimum and maximum partitions (written disk IO [min, max]). The Sort operator does not display the number of files written to disks, and displays disks only when displaying sorting methods.

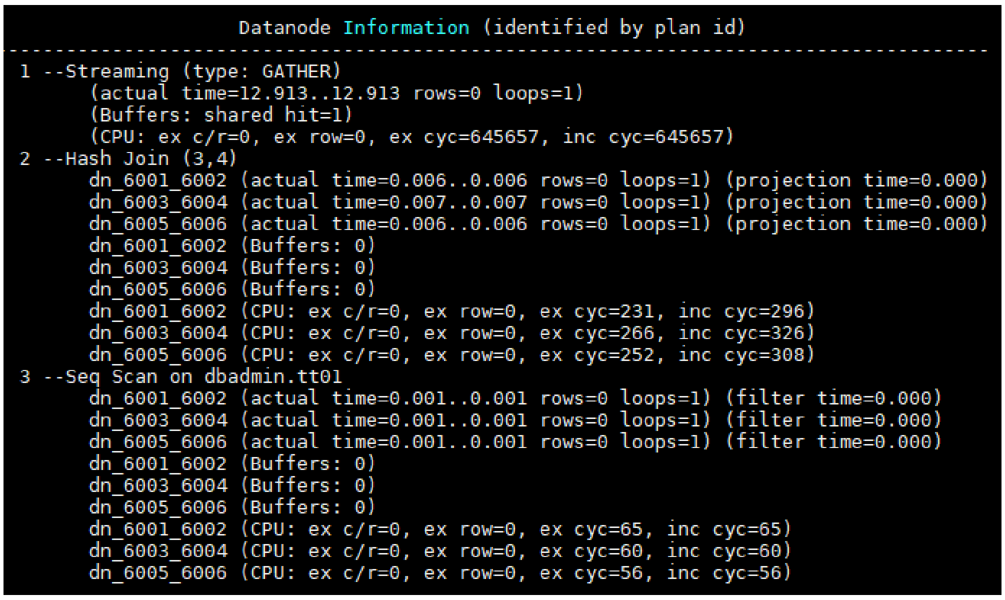
5. Targetlist Information (identified by plan id)



```
  Targetlist Information (identified by plan id)
-----------------------------------------------------
  1 --Streaming (type: GATHER)
        Output: tt01.c1, tt01.c2, tt02.c1, tt02.c2
        Node/s: All datanodes
  2 --Hash Join (3,4)
        Output: tt01.c1, tt01.c2, tt02.c1, tt02.c2
  3 --Seq Scan on dbadmin.tt01
        Output: tt01.c1, tt01.c2
        Distribute Key: tt01.c1
  4 --Hash
        Output: tt02.c1, tt02.c2
  5 --Seq Scan on dbadmin.tt02
        Output: tt02.c1, tt02.c2
        Distribute Key: tt02.c2
```

This part displays the output target column information of each operator.

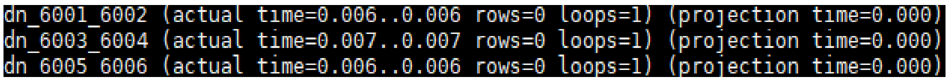
6. DataNode Information (identified by plan id)



```
                     Datanode Information (identified by plan id)
-------------------------------------------------------------------------------------
 1 --Streaming (type: GATHER)
       (actual time=12.913..12.913 rows=0 loops=1)
       (Buffers: shared hit=1)
       (CPU: ex c/r=0, ex row=0, ex cyc=645657, inc cyc=645657)
 2 --Hash Join (3,4)
       dn_6001_6002 (actual time=0.006..0.006 rows=0 loops=1) (projection time=0.000)
       dn_6003_6004 (actual time=0.007..0.007 rows=0 loops=1) (projection time=0.000)
       dn_6005_6006 (actual time=0.006..0.006 rows=0 loops=1) (projection time=0.000)
       dn_6001_6002 (Buffers: 0)
       dn_6003_6004 (Buffers: 0)
       dn_6005_6006 (Buffers: 0)
       dn_6001_6002 (CPU: ex c/r=0, ex row=0, ex cyc=231, inc cyc=296)
       dn_6003_6004 (CPU: ex c/r=0, ex row=0, ex cyc=266, inc cyc=326)
       dn_6005_6006 (CPU: ex c/r=0, ex row=0, ex cyc=252, inc cyc=308)
 3 --Seq Scan on dbadmin.tt01
       dn_6001_6002 (actual time=0.001..0.001 rows=0 loops=1) (filter time=0.000)
       dn_6003_6004 (actual time=0.001..0.001 rows=0 loops=1) (filter time=0.000)
       dn_6005_6006 (actual time=0.001..0.001 rows=0 loops=1) (filter time=0.000)
       dn_6001_6002 (Buffers: 0)
       dn_6003_6004 (Buffers: 0)
       dn_6005_6006 (Buffers: 0)
       dn_6001_6002 (CPU: ex c/r=0, ex row=0, ex cyc=65, inc cyc=65)
       dn_6003_6004 (CPU: ex c/r=0, ex row=0, ex cyc=60, inc cyc=60)
       dn_6005_6006 (CPU: ex c/r=0, ex row=0, ex cyc=56, inc cyc=56)
```

This part displays the execution time of each operator (including the execution time of filtering and projection, if any), CPU usage, and buffer usage.

– Operator execution information

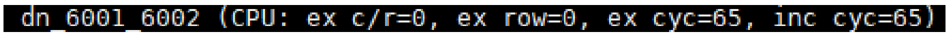


```
dn_6001_6002 (actual time=0.006..0.006 rows=0 loops=1) (projection time=0.000)
dn_6003_6004 (actual time=0.007..0.007 rows=0 loops=1) (projection time=0.000)
dn_6005_6006 (actual time=0.006..0.006 rows=0 loops=1) (projection time=0.000)
```

The execution information of each operator consists of three parts:

- **dn_6001_6002**/**dn_6003_6004** indicates the information about the execution node. The information in the brackets is the actual execution information.
- **actual time** indicates the actual execution time. The first number indicates the duration from the time when the operator is executed to the time when the first data record is output. The second number indicates the total execution time of all data records.
- **rows** indicates the number of output data rows of the operator.
- **loops** indicates the number of execution times of the operator. Note that for a partitioned table, scan on each partition is counted as a scan. Scan on a new partition is counted as a new scan.

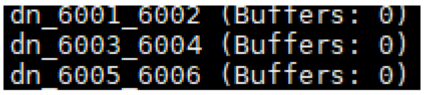
– CPU information



```
 dn_6001_6002 (CPU: ex c/r=0, ex row=0, ex cyc=65, inc cyc=65)
```

Each operator execution process has CPU information. **cyc** indicates the number of CPU cycles, and **ex cyc** indicates the number of cycles of the current operator, excluding its subnodes. **inc cyc** indicates the number of cycles, including subnodes, **ex row** indicates the number of data rows output by the current operator, and **ex c/r** indicates the mean of **ex cyc** and **ex row**.

– Buffer information

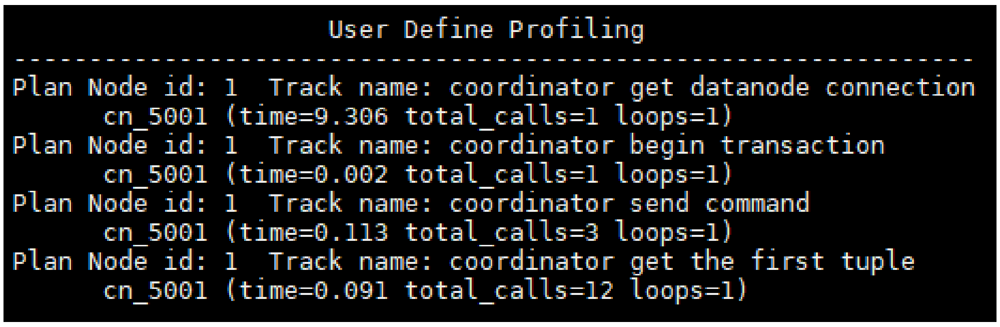


```
dn_6001_6002 (Buffers: 0)
dn_6003_6004 (Buffers: 0)
dn_6005_6006 (Buffers: 0)
```

Buffers indicates the buffer information, including the read and write operations on shared blocks and temporary blocks.

Shared blocks contain tables and indexes, and temporary blocks are disk blocks used in sorting and materialization. The number of blocks displayed on the upper-layer node contains the number of blocks used by all its subnodes.

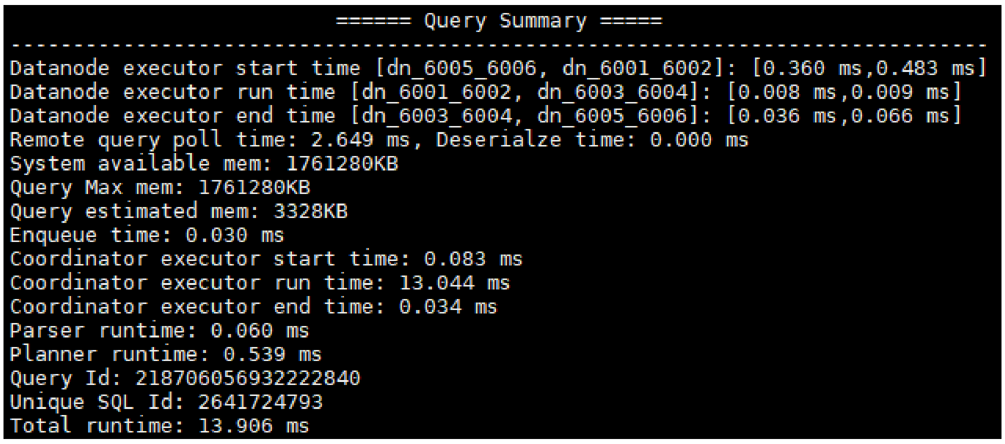
7. User Define Profiling



```
                    User Define Profiling
-------------------------------------------------------------
Plan Node id: 1  Track name: coordinator get datanode connection
      cn_5001 (time=9.306 total_calls=1 loops=1)
Plan Node id: 1  Track name: coordinator begin transaction
      cn_5001 (time=0.002 total_calls=1 loops=1)
Plan Node id: 1  Track name: coordinator send command
      cn_5001 (time=0.113 total_calls=3 loops=1)
Plan Node id: 1  Track name: coordinator get the first tuple
      cn_5001 (time=0.091 total_calls=12 loops=1)
```

User-defined information, including the time when CNs and DNs are connected, the time when DNs are connected, and some execution information at the storage layer.

8. Query Summary



```
                          ====== Query Summary =====
-------------------------------------------------------------------------------
Datanode executor start time [dn_6005_6006, dn_6001_6002]: [0.360 ms,0.483 ms]
Datanode executor run time [dn_6001_6002, dn_6003_6004]: [0.008 ms,0.009 ms]
Datanode executor end time [dn_6003_6004, dn_6005_6006]: [0.036 ms,0.066 ms]
Remote query poll time: 2.649 ms, Deserialze time: 0.000 ms
System available mem: 1761280KB
Query Max mem: 1761280KB
Query estimated mem: 3328KB
Enqueue time: 0.030 ms
Coordinator executor start time: 0.083 ms
Coordinator executor run time: 13.044 ms
Coordinator executor end time: 0.034 ms
Parser runtime: 0.060 ms
Planner runtime: 0.539 ms
Query Id: 218706056932222840
Unique SQL Id: 2641724793
Total runtime: 13.906 ms
```

The total execution time and network traffic, including the maximum and minimum execution time in the initialization and end phases on each DN, initialization, execution, and time in the end phase on each CN, and the system available memory during the current statement execution, and statement estimation memory information.

– DataNode executor start time: start time of the DN executor. The format is [min_node_name, max_node_name]: [min_time, max_time].
– DataNode executor run time: running time of the DN executor. The format is [min_node_name, max_node_name]: [min_time, max_time].
– DataNode executor end time: end time of the DN executor. The format is [min_node_name, max_node_name]: [min_time, max_time].
– **Remote query poll time**: poll waiting time for receiving results
– System available mem: available system memory
– Query Max mem: maximum query memory.
– Enqueue time: enqueuing time

- Coordinator executor start time: start time of the CN executor
- Coordinator executor run time: CN executor running time
- Coordinator executor end time: end time of the CN executor
- Parser runtime: parser running time
- Planner runtime: optimizer execution time
- Network traffic, or, the amount of data sent by the stream operator
- Query Id: query ID.
- Unique SQL ID: constraint SQL ID
- Total runtime: total execution time

NOTICE

- The difference between A-rows and E-rows shows the deviation between the optimizer estimation and actual execution. Generally, if the deviation is large, the plan generated by the optimizer cannot be trusted, and you need to modify the deviation value.
- If the difference of the A-time values is large, it indicates that the operator computing skew (difference between execution time on DNs) is large and that manual performance tuning is required. Generally, for two adjacent operators, the execution time of the upper-layer operator includes that of the lower-layer operator. However, if the upper-layer operator is a stream operator, its execution time may be less than that of the lower-layer operator, as there is no driving relationship between threads.
- **Max Query Peak Memory** is often used to estimate the consumed memory of SQL statements, and is also used as an important basis for setting a memory parameter during SQL statement optimization. Generally, the output from **EXPLAIN ANALYZE** or **EXPLAIN PERFORMANCE** is provided for the input for further optimization.

13.4.3 Execution Plan Operator

Operator Introduction

In an SQL execution plan, each step indicates a database operator, also called an execution operator. In GaussDB(DWS), operators are the building blocks of data processing. By combining them effectively and optimizing their sequence and execution, you can significantly improve data processing efficiency.

GaussDB(DWS) operators are classified into scan operators, control operators, materialization operators, join operators, and other operators.

Scan Operators

A scan operator scans data in a table, processing one tuple at a time for the upper-layer node. It operates at the leaf node of the query plan tree and can scan tables, result sets, linked lists, and subquery results. The following table lists common scan operators.

Table 13-8 Scan operators

Operator	Description	Scenario
SeqScan	Sequential scanning	It is a basic operator used to scan physical tables in sequence, not an index-assisted scan.
IndexScan	Index scanning	Indexes are created for the attributes involved in selection conditions.
IndexOnlyScan	Obtaining a tuple from an index	The index column completely overwrites the result set column.
BitmapScan(BitmapIndexScan, BitmapHeapScan)	Obtaining a tuple using a bitmap	BitmapIndexScan uses indexes for attributes to scan data and returns a bitmap. BitmapHeapScan then uses this bitmap to retrieve tuples.
TidScan	Obtaining a tuple by tuple tid	1. WHERE conditions(like CTID = tid or CTID IN (tid1, tid2, ...)) ; 2. UPDATE/DELETE ... WHERE CURRENT OF cursor;
SubqueryScan	Subquery scanning	Another query plan tree (subplan) is used as the scanning object to scan tuples.
FunctionScan	Function scanning	FROM function_name
ValuesScan	Values linked list scanning	It scans the given tuple set in VALUES clauses.
ForeignScan	External table scanning	It queries external tables.
CteScan	CTE table scanning	It scans the subquery defined by the WITH clause in the SELECT query.

Join Operators

In relational algebra, a join operation is equivalent to a join operator. Take a simple example: joining two tables, t1 and t2. There are several types of joins, including inner join, left join, right join, full join, semi join, and anti join. These joins can be implemented using three methods: Nestloop, HashJoin, and MergeJoin.

Table 13-9 Join operators

Operator	Description	Scenario	Implementation Feature
NestLoop	Nested loop join, which is a brute force approach. It scans the inner table for each row.	Inner Join, Left Outer Join, Semi Join, Anti Join	It is used for queries that have a smaller subset connected. In a nested loop, the foreign table drives the internal table. Each row returned by the foreign table is retrieved from the internal table to find the matched row. Therefore, the result set returned by the entire query cannot be greater than 10,000. The table with a smaller subset returned is used as the foreign table. It is recommended that indexes be created for the join fields in the internal table.
MergeJoin	A merge join on ordered input sorts both the inner and outer tables, identifies the first and last matching rows, and then joins tuples at a time. Equi-join.	Inner Join, Left Outer Join, Right Outer Join, Full Outer Join, Semi Join, Anti Join	In a merge join, data in the two joined tables is sorted by join columns. Then, data is extracted from the two tables to a sorted table for matching. A merge join requires more resources for sorting and its performance 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.
(Sonic) Hash Join	Hash join: The inner and outer tables use the join column's hash value to create a hash table. Matching values are then stored in the same bucket. The two ends of an equal join must be of the same type and support hash.	Inner Join, Left Outer Join, Right Outer Join, Full Outer Join, Semi Join, Anti Join	A hash Join is used for large tables. The optimizer creates a hash table in memory using the join key and the smaller table. It then scans the larger table and uses the hash table to quickly identify matching rows. While Sonic and non-Sonic hash joins have different internal structures, this does not impact the final result set.

Materialization Operators

Materialization operators are a class of nodes that can cache tuples. During execution, many extended physical operations can be performed only after all tuples are obtained, such as aggregation function operations and sorting without indexes. Materialization operators can cache all the tuples.

Table 13-10 Materialization operators

Operator	Description	Scenario
Material	Materialization	Caches the subnode result.
Sort	Sorting	ORDER BY clause, which is used for join, group, and set operations and works with Unique.
Group	Grouping	GROUP BY clause.
Agg	Executes aggregate functions.	1. Aggregate functions such as COUNT, SUM, AVG, MAX, and MIN. 2. DISTINCT clause. 3. UNION deduplication. 4. GROUP BY clause.
WindowAgg	Window functions	WINDOW clause.
Unique	Deduplication (with sorted lower-layer data)	1. DISTINCT clause. 2. UNION deduplication.
Hash	HashJoin auxiliary node	Constructs a hash table and use it together with HashJoin.
SetOp	Processing set operations	INTERSECT/INTERSECT ALL, EXCEPT/EXCEPT ALL
LockRows	Processing row-level locks	SELECT ... FOR SHARE/UPDATE

Control Operators

Control operators are a type of node that handles exceptional scenarios and executes custom workflows.

Table 13-11 Control operators

Operator	Description	Scenario
Result	Performing calculation directly	1. Table scanning is not included. 2. The INSERT statement contains only one VALUES clause.
ModifyTable	INSERT/UPDATE/ DELETE upper-layer node	INSERT/UPDATE/DELETE
Append	Appending	1. UNION(ALL). 2. Table inheritance.
MergeAppend	Appending (ordered input)	1. UNION(ALL). 2. Table inheritance.
RecursiveUnion	Processing the UNION subquery defined recursively in the WITH clause	WITH RECURSIVE... SELECT... statement.
BitmapAnd	Bitmap logical AND operation	BitmapScan for multi-dimensional index scanning.
BitmapOr	Bitmap logical OR operation	BitmapScan for multi-dimensional index scanning.
Limit	Processing the LIMIT clause	OFFSET ... LIMIT ...

Other Operators

Other operators include Stream and RemoteQuery. There are three types of Stream operators: Gather stream, Broadcast stream, and Redistribute stream.

- Gather stream: Each source node sends its data to the target node for aggregation.
- Broadcast stream: A source node sends its data to N target nodes for calculation.
- Redistribute stream: Each source node calculates the hash value of its data based on the join condition, distributes the data based on the hash value, and sends the data to the corresponding target node.

Table 13-12 Other Operators

Operator	Description	Scenario
Stream	Multi-node data exchange	When a distributed query plan is executed, data is exchanged between nodes.

Operator	Description	Scenario
Partition Iterator	Partition iterator	Scans partitioned tables and iteratively scans each partition.
RowToVec	Rows-to-column conversion	Hybrid row-column.
DfsScan / DfsIndexScan	HDFS table (index) scanning	HDFS table scanning.

13.4.4 SQL 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 statistics are unavailable or out of date, the execution plan may seriously deteriorate, leading to low performance. According to past experience, about 10% performance problem occurred because no statistics are collected. For details, see **Updating Statistics**.

Step 2 **Review and modify the table definition.**

Step 3 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. The query rewriting method is universal in all databases. **SQL Statement Rewriting Rules** describes several optimization methods by rewriting SQL statements.

Step 4 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** command to view the execution plan and then locate the fault. If the SQL statement has been executed, run the **EXPLAIN ANALYZE** or **EXPLAIN PERFORMANCE** command to check the execution plan and actual running situation and then accurately locate the fault. For details about the execution plan, see **SQL Execution Plan**.

Step 5 For details about **EXPLAIN** or **EXPLAIN PERFORMANCE**, the reason why SQL statements are slowly located, and how to solve this problem, see **Advanced SQL Tuning**.

Step 6 Specify a join order; join, stream, or scan operations; number of rows in a result; or redistribution skew information to optimize an execution plan, improving query performance. For details, see **Hint-based Tuning**.

Step 7 To maintain high database performance, you are advised to perform **Routinely Maintaining Tables** and **Routinely Recreating an Index**.

Step 8 (Optional) Improve performance by using operators if resources are sufficient in GaussDB(DWS). For details, see **SMP Parallel Execution**.

----End

13.4.5 Updating Statistics

In a database, statistics indicate the source data of a plan generated by a planner. If statistics are unavailable or out of date, the execution plan may seriously deteriorate, leading to low performance.

Context

The **ANALYZE** statement collects statistic about table contents in databases, which will be stored in the system table **PG_STATISTIC**. Then, the query optimizer uses the statistics to work out the most efficient execution plan.

After executing batch insertion 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.

Generating Statistics

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:

```
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.
- Use **EXPLAIN** to show the execution plan of each SQL statement. If **rows=10** (the default value, probably indicating the table has not been analyzed) is displayed in the **SEQ SCAN** output of a table, run the **ANALYZE** statement for this table.

Improving the Quality of Statistics

ANALYZE samples data from a table based on the random sampling algorithm and calculates table data features based on the samples. The number of samples can be specified by the **default_statistics_target** parameter. The value of **default_statistics_target** ranges from -100 to 10000, and the default value is 100.

- If the value of **default_statistics_target** is greater than **0**, the number of samples is 300 x **default_statistics_target**. This means a larger value of **default_statistics_target** indicates a larger number of samples, larger memory space occupied by samples, and longer time required for calculating statistics.
- If the value of **default_statistics_target** is smaller than **0**, the number of samples is **default_statistics_target**/100 x Total number of rows in the table. A smaller value of **default_statistics_target** indicates a larger number of samples. If the value of **default_statistics_target** is smaller than **0**, the sampled data is written to the disk. In this case, the samples do not occupy memory. However, the calculation still takes a long time because the sample size is too large.

  When **default_statistics_target** < 0, the actual number of samples is **default_statistics_target**/100 x Total number of rows in the table. Therefore, this sampling mode is also called percentage sampling.

Automatic Statistics Collection

When the parameter **autoanalyze** is enabled, if the query statement reaches the optimizer and finds that there are no statistics, statistics collection will be automatically triggered to meet the optimizer's requirements.

Note: Automatic statistics collection is triggered only for complex query SQL statements that are sensitive to statistics (such as multi-table association). Simple queries (such as single-point query and single-table aggregation) do not trigger automatic statistics collection.

13.4.6 Reviewing and Modifying a Table Definition

In a distributed framework, data is distributed on DNs. Data on one or more DNs is stored on a physical storage device. To properly define a table, you must:

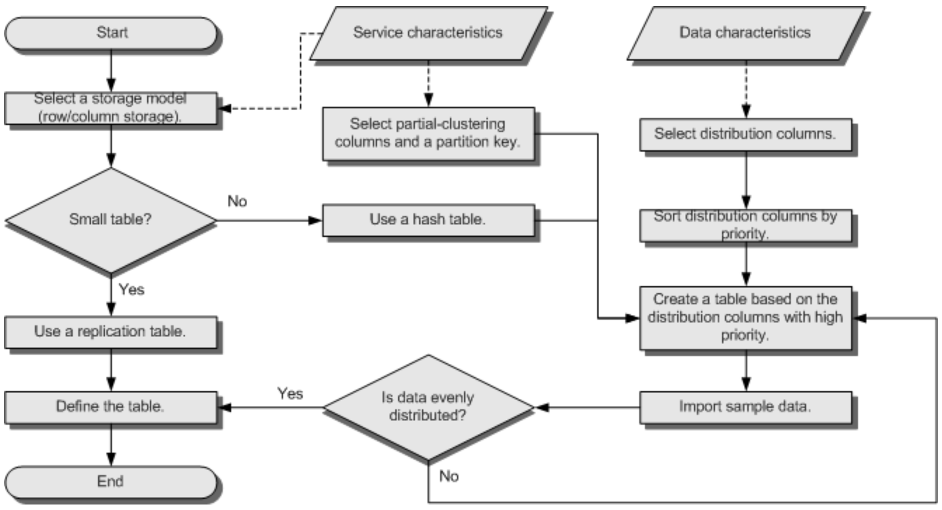
1. **Evenly distribute data on each DN** to avoid the available capacity decrease of a cluster caused by insufficient storage space of the storage device

associated with a DN. Specifically, select a proper distribution key to avoid data skew.

2. **Evenly assign table scanning tasks on each DN** to avoid that a DN is overloaded by the table scanning tasks. Specifically, do not select columns in the equivalent filter of a base table as the distribution key.
3. **Reduce the data volume scanned** by using the partition pruning mechanism.
4. **Avoid the use of random I/O** by using clustering or partial clustering.
5. **Avoid data shuffle** to reduce the network pressure by selecting the **join-condition** column or **group by** column as the distribution column.

The distribution column is the core for defining a table. **Figure 13-5** shows the procedure of defining a table. The table definition is created during the database design and is reviewed and modified during SQL tuning.

Figure 13-5 Defining a table


Start
Service characteristics
Data characteristics
Select a storage model (row/column storage).
Select partial-clustering columns and a partition key.
Select distribution columns.
Small table?
No
Use a hash table.
Sort distribution columns by priority.
Yes
Use a replication table.
Create a table based on the distribution columns with high priority.
Define the table.
Yes
Is data evenly distributed?
Import sample data.
End
No

For details about how to review and modify table definitions, see **Table Optimization Practices**.

13.4.7 Advanced SQL Tuning

13.4.7.1 SQL Self-Diagnosis

Performance issues may occur when you run the **INSERT/UPDATE/DELETE/SELECT/MERGE INTO** or **CREATE TABLE AS** statement. The product supports automatic performance diagnosis and saves related diagnosis information to **Real-time Top SQL**. When **enable_resource_track** is set to **on**, the diagnosis information is dumped to **Historical Top SQL**. You can query the **warning** column in the **GS_WLM_SESSION_STATISTICS**, **GS_WLM_SESSION_HISTORY**, and **GS_WLM_SESSION_INFO** views to obtain reference information for performance tuning.

- Alarms that can trigger SQL self-diagnosis depend on the settings of **resource_track_level**.

  When **resource_track_level** is set to **query**, you can diagnose alarms such as uncollected multi-column/single-column statistics, unpruned partitions, and failure of pushing down SQL statements. When **resource_track_level** is set to **perf** or **operator**, all alarms can be diagnosed.

- 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.
- When **EXPLAIN PERFORMANCE** or **EXPLAIN VERBOSE** is executed, SQL self-diagnosis information, except the ones without multi-column statistics, will be generated. For details, see **SQL Execution Plan**.

Alarms Related to SQL Execution Performance

Currently, the following alarms on performance issues will be reported:

1. Statistics of a single column or multiple columns are not collected.

   If statistics of a single column or multiple columns are not collected, an alarm is reported. To handle this alarm, you are advised to perform **ANALYZE** on related tables. For details, see **Updating Statistics** and **Optimizing Statistics**.

   If no statistics are collected for the OBS foreign table and HDFS foreign table in the query statement, an alarm indicating that statistics are not collected will be reported. Because the **ANALYZE** performance of the OBS foreign table and HDFS foreign table is poor, you are not advised to perform **ANALYZE** on these tables. Instead, you are advised to use the **ALTER FOREIGN TABLE** syntax to modify the **totalrows** attribute of the foreign table to correct the estimated number of rows.

   Example alarms:

   The 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)
   ```

   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)
         schema_test.t5((c1,c2))
   ```

2. Partitions are not pruned.

   When a partitioned table is queried, the partition is pruned based on the constraints on the partition key to improve the query performance. However, the partition table may not be pruned due to improper constraints, deteriorating the query performance. For details, see **Case: Rewriting SQL Statements and Eliminating Prune Interference**.

3. SQL statements are not pushed down.

   The cause details are displayed in the alarms. For details, see **Optimizing Statement Pushdown**.

   The potential causes for the pushdown failure are as follows:

   - Caused by functions

     The function name is displayed in the diagnosis information. Function pushdown is determined by the **shippable** attribute of the function. For details, see the **CREATE FUNCTION** syntax.

   - Caused by syntax

     The diagnosis information displays the syntax that causes the pushdown failure. For example, if the statement contains the **With Recursive**, **Distinct On**, or **row** expression and the return value is of the record type, an alarm is reported, indicating that the syntax does not support pushdown.

   Example alarms:

```
SQL is not plan-shipping
        "enable_stream_operator" is off

SQL is not plan-shipping
        "Distinct On" can not be shipped

SQL is not plan-shipping
        "v_test_unshipping_log" is VIEW that will be treated as Record type can't be shipped
```

4. Vectorized plans are not supported.

   For SQL statements that cannot use vectorized plans, detailed reasons why vectorized plans cannot be used are reported.

   Common reasons are as follows:

   - The target column contains functions whose return type is a set.
   - The target column or query condition, the distribution key of the Stream operator, and the **Limit** and **Offset** clauses contain expressions that cannot be vectorized (such as geospatial types, array expressions, Row expressions, XML expressions, and functions whose parameters or return values contain the refcursor type).
   - The **Group By** clause contains an array-equivalent judgment statement.
   - **GC_FDW** and **LOG_FDW** do not support vectorization.
   - The plan contains operators such as Cte Scan, Recursive Union, Merge Append, and Lock Rows.

   Example alarms:

```
SQL is un-vectorized
         Function regexp_split_to_table that returns set is un-vectorized

SQL is un-vectorized
         Array expression is un-vectorized

SQL is un-vectorized
         Function array_agg is un-vectorized

SQL is un-vectorized
         RecursiveUnion is un-vectorized
```

5. In a hash join, the larger table is used as the inner table.

   An alarm will be reported if the number of rows in the inner table reaches or exceeds 10 times of that in the foreign table, more than 100,000 inner-table rows are processed on each DN in average, and data has been flushed to disks. You can check the **query_plan** column in **GS_WLM_SESSION_HISTORY** to check whether hash joins are used. In this scenario, you need to adjust the sequence of the HashJoin internal and foreign tables. For details, see **Join Order Hints**.

Example alarm:

```
Execute diagnostic information
PlanNode[7] Large Table is INNER in HashJoin "Vector Hash Aggregate"
```

In the preceding command, **7** indicates the operator whose ID is **7** in the **query_plan** column.

6. **nestloop** is used in a large-table equivalent join.

   An alarm will be reported if nested loop is used in an equivalent join where more than 100,000 larger-table rows are processed on each DN in average. You can check the **query_plan** column of **GS_WLM_SESSION_HISTORY** to see if nested loop is used. In this scenario, you need to adjust the table join mode and disable the NestLoop join mode between the current internal and foreign tables. For details, see **Join Operation Hints**.

   Example alarm:

```
Execute diagnostic information
      PlanNode[5] Large Table with Equal-Condition use Nestloop"Nested Loop"
```

7. A large table is broadcasted.

   An alarm will be reported if more than 100 thousand of rows are broadcasted on each DN in average. In this scenario, the broadcast operation of the Broadcast lower-layer operator needs to be disabled. For details, see **Stream Operation Hints**.

   Example alarm:

```
Execute diagnostic information
      PlanNode[5] Large Table in Broadcast "Streaming(type: BROADCAST dop: 1/2)"
```

8. Data skew occurs.

   An alarm will be reported if the number of rows processed on any DN exceeds 100 thousand, and the number of rows processed on a DN reaches or exceeds 10 times of that processed on another DN. Generally, this alarm is generated due to storage layer skew or computing layer skew. For details, see **Optimizing Data Skew**.

   Example alarm:

```
Execute diagnostic information
     PlanNode[6] DataSkew:"Seq Scan", min_dn_tuples:0, max_dn_tuples:524288
```

9. The index is improper.

   During base table scanning, an alarm is reported if the following conditions are met:

   – For row-store tables:

     ▪ When the index scanning is used, the ratio of the number of output lines to the number of scanned lines is greater than 1/1000 and the number of output lines is greater than 10,000.

     ▪ When sequential scanning is used, the number of output lines to the number of scanned lines is less than 1/1000, the number of output lines is less than or equal to 10,000, and the number of scanned lines is greater than 10,000.

   – For column-store tables:

     ▪ When the index scanning is used, the ratio of the number of output lines to the number of scanned lines is greater than 1/10000 and the number of output lines is greater than 100.

  - ▪ When sequential scanning is used, the number of output lines to the number of scanned lines is less than 1/10,000, the number of output lines is less than or equal to 100, and the number of scanned lines is greater than 10,000.

  For details, see **Optimizing Operators**. You can also refer to **Case: Creating an Appropriate Index** and **Case: Setting Partial Cluster Keys**.

  Example alarms:

```
Execute diagnostic information
       PlanNode[4] Indexscan is not properly used:"Index Only Scan", output:524288, filtered:0,
rate:1.00000
       PlanNode[5] Indexscan is ought to be used:"Seq Scan", output:1, filtered:524288, rate:0.00000
```

  The diagnosis result is only a suggestion for the current SQL statement. You are advised to create an index only for frequently used filter criteria.

10. Estimation is inaccurate.

An alarm will be reported if the maximum number or the estimated maximum number of rows processed on a DN is over 100,000, and the larger number reaches or exceeds 10 times of the smaller one. In this scenario, you can refer to **Rows Hints** to correct the estimation on the number of rows, so that the optimizer can re-design the execution plan based on the correct number.

Example alarm:

```
Execute diagnostic information
       PlanNode[5] Inaccurate Estimation-Rows: "Hash Join" A-Rows:0, E-Rows:52488
```

Constraints

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.
3. For alarms about data skew and inaccurate estimation, only alarms on the lower-layer nodes in a plan tree will be reported. This is because the same alarms on the upper-level nodes may be triggered by problems on the lower-layer nodes. For example, if data skew occurs on the **Scan** node, data skew may also occur in operators (for example, **Hashagg**) at the upper layer.

13.4.7.2 Optimizing Statement Pushdown

Statement Pushdown

Currently, the GaussDB(DWS) optimizer can use three methods to develop statement execution policies in the distributed framework: generating a statement pushdown plan, a distributed execution plan, or a distributed execution plan for sending statements.

- A statement pushdown plan pushes query statements from a CN down to DNs for execution and returns the execution results to the CN.
- In a distributed execution plan, a CN compiles and optimizes query statements, generates a plan tree, and then sends the plan tree to DNs for

execution. After the statements have been executed, execution results will be returned to the CN.

- A distributed execution plan for sending statements pushes queries that can be pushed down (mostly base table scanning statements) to DNs for execution. Then, the plan obtains the intermediate results and sends them to the CN, on which the remaining queries are to be executed.

The third policy sends many intermediate results from the DNs to a CN for further execution. In this case, the CN performance bottleneck (in bandwidth, storage, and computing) is caused by statements that cannot be pushed down to DNs. Therefore, you are not advised to use the query statements that only the third policy is applicable to.

Statements cannot be pushed down to DNs if they have **Functions That Do Not Support Pushdown** or **Syntax That Does Not Support Pushdown**. Generally, you can rewrite the execution statements to solve the problem.

Viewing Whether the Execution Plan Has Been Pushed Down to DNs

Perform the following procedure to quickly determine whether the execution plan can be pushed down to DNs:

Step 1 Set the GUC parameter to **off** to use the distributed framework policy for the query optimizer.

```
SET enable_fast_query_shipping = off;
```

Step 2 View the execution plan.

If the execution plan contains Data Node Scan, the SQL statements cannot be pushed down to DNs. If the execution plan contains Streaming, the SQL statements can be pushed down to DNs.

For example:

```
select
count(ss.ss_sold_date_sk order by ss.ss_sold_date_sk)c1
from store_sales ss, store_returns sr
where
sr.sr_customer_sk = ss.ss_customer_sk;
```

The execution plan is as follows, which indicates that the SQL statement cannot be pushed down.

```
                              QUERY PLAN
--------------------------------------------------------------------------
Aggregate
->  Hash Join
Hash Cond: (ss.ss_customer_sk = sr.sr_customer_sk)
->  Data Node Scan on store_sales "_REMOTE_TABLE_QUERY_"
Node/s: All datanodes
->  Hash
->  Data Node Scan on store_returns "_REMOTE_TABLE_QUERY_"
Node/s: All datanodes
(8 rows)
```

----End

Syntax That Does Not Support Pushdown

SQL syntax that does not support pushdown is described using the following table definition examples:

```
CREATE TABLE CUSTOMER1
(
    C_CUSTKEY     BIGINT NOT NULL
  , C_NAME        VARCHAR(25) NOT NULL
  , C_ADDRESS     VARCHAR(40) NOT NULL
  , C_NATIONKEY   INT NOT NULL
  , C_PHONE       CHAR(15) NOT NULL
  , C_ACCTBAL     DECIMAL(15,2)   NOT NULL
  , C_MKTSEGMENT  CHAR(10) NOT NULL
  , C_COMMENT     VARCHAR(117) NOT NULL
)
DISTRIBUTE BY hash(C_CUSTKEY);
CREATE TABLE test_stream(a int, b float);--float does not support redistribution.
CREATE TABLE sal_emp ( c1 integer[] ) DISTRIBUTE BY replication;
```

- The **returning** statement cannot be pushed down.

```
explain update customer1 set C_NAME = 'a' returning c_name;
                                QUERY PLAN
------------------------------------------------------------------
 Update on customer1  (cost=0.00..0.00 rows=30 width=187)
   Node/s: All datanodes
   Node expr: c_custkey
   ->  Data Node Scan on customer1 "_REMOTE_TABLE_QUERY_"  (cost=0.00..0.00 rows=30 width=187)
         Node/s: All datanodes
(5 rows)
```

- If columns in **count(distinct expr)** do not support redistribution, they do not support pushdown.

```
explain verbose select count(distinct b) from test_stream;
                                           QUERY PLAN
------------------------------------------------------------------ Aggregate  (cost=2.50..2.51 rows=1 width=8)
   Output: count(DISTINCT test_stream.b)
   ->  Data Node Scan on test_stream "_REMOTE_TABLE_QUERY_"  (cost=0.00..0.00 rows=30 width=8)
         Output: test_stream.b
         Node/s: All datanodes
         Remote query: SELECT b FROM ONLY public.test_stream WHERE true
(6 rows)
```

- Statements using **distinct on** cannot be pushed down.

```
explain verbose select distinct on (c_custkey) c_custkey from customer1 order by c_custkey;
                                           QUERY PLAN
------------------------------------------------------------------ Unique  (cost=49.83..54.83 rows=30 width=8)
   Output: customer1.c_custkey
   ->  Sort  (cost=49.83..52.33 rows=30 width=8)
         Output: customer1.c_custkey
         Sort Key: customer1.c_custkey
         ->  Data Node Scan on customer1 "_REMOTE_TABLE_QUERY_"  (cost=0.00..0.00 rows=30
width=8)
               Output: customer1.c_custkey
               Node/s: All datanodes
               Remote query: SELECT c_custkey FROM ONLY public.customer1 WHERE true
(9 rows)
```

- In a statement using **FULL JOIN**, if the column specified using **JOIN** does not support redistribution, the statement does not support pushdown.

```
explain select * from test_stream t1 full join test_stream t2 on t1.a=t2.b;
                                           QUERY PLAN
------------------------------------------------------------------ Hash Full Join  (cost=0.38..0.82 rows=30
width=24)
   Hash Cond: ((t1.a)::double precision = t2.b)
   ->  Data Node Scan on test_stream "_REMOTE_TABLE_QUERY_"  (cost=0.00..0.00 rows=30 width=12)
         Node/s: All datanodes
   ->  Hash  (cost=0.00..0.00 rows=30 width=12)
         ->  Data Node Scan on test_stream "_REMOTE_TABLE_QUERY_"  (cost=0.00..0.00 rows=30
width=12)
               Node/s: All datanodes
(7 rows)
```

- Does not support array expression pushdown.

```
explain verbose select array[c_custkey,1] from customer1 order by c_custkey;
```

```
                          QUERY PLAN
------------------------------------------------------------------ Sort  (cost=49.83..52.33 rows=30 width=8)
   Output: (ARRAY[customer1.c_custkey, 1::bigint]), customer1.c_custkey
   Sort Key: customer1.c_custkey
   ->  Data Node Scan on "__REMOTE_SORT_QUERY__"  (cost=0.00..0.00 rows=30 width=8)
         Output: (ARRAY[customer1.c_custkey, 1::bigint]), customer1.c_custkey
         Node/s: All datanodes
         Remote query: SELECT ARRAY[c_custkey, 1::bigint], c_custkey FROM ONLY public.customer1
WHERE true ORDER BY 2
(7 rows)
```

- The following table describes the scenarios where a statement containing **WITH RECURSIVE** cannot be pushed down in the current version, as well as the causes.

No.	Scenario	Cause of Not Supporting Pushdown
1	The query contains foreign tables or HDFS tables.	LOG: SQL can't be shipped, reason: RecursiveUnion contains HDFS Table or ForeignScan is not shippable (In this table, **LOG** describes the cause of not supporting pushdown.) In the current version, queries containing foreign tables or HDFS tables do not support pushdown.
2	Multiple Node Groups	LOG: SQL can't be shipped, reason: With-Recursive under multi-nodegroup scenario is not shippable In the current version, pushdown is supported only when all base tables are stored and computed in the same Node Group.

3	`WITH recursive t_result AS (`
	`SELECT dm,sj_dm,name,1 as level`
	`FROM test_rec_part`
	`WHERE sj_dm > 10`
	`UNION`
	`SELECT t2.dm,t2.sj_dm,t2.name||' > '||`
	`t1.name,t1.level+1`
	`FROM t_result t1`
	`JOIN test_rec_part t2 ON t2.sj_dm = t1.dm`
	`)`
	`SELECT * FROM t_result t;`
		LOG: SQL can't be shipped, reason: With-Recursive does not contain "ALL" to bind recursive & none-recursive branches
		
		ALL is not used for **UNION**. In this case, the return result is deduplicated.

No.	Scenario	Cause of Not Supporting Pushdown

4	`WITH RECURSIVE x(id) AS`
	`(`
	`select count(1) from pg_class where oid=1247`
	`UNION ALL`
	`SELECT id+1 FROM x WHERE id < 5`
	`), y(id) AS`
	`(`
	`select count(1) from pg_class where oid=1247`
	`UNION ALL`
	`SELECT id+1 FROM x WHERE id < 10`
	`)`
	`SELECT y.*, x.* FROM y LEFT JOIN x USING (id) ORDER BY 1;`
		LOG: SQL can't be shipped, reason: With-Recursive contains system table is not shippable
		
		A base table contains the system catalog.

5	`WITH RECURSIVE t(n) AS (`
	`VALUES (1)`
	`UNION ALL`
	`SELECT n+1 FROM t WHERE n < 100`
	`)`
	`SELECT sum(n) FROM t;`
		LOG: SQL can't be shipped, reason: With-Recursive contains only values rte is not shippable
		
		Only **VALUES** is used for scanning base tables. In this case, the statement can be executed on the CN, and DNs are unnecessary.

6	`select  a.ID,a.Name,`
	`(`
	`with recursive cte as (`
	`select ID, PID, NAME from b where b.ID = 1`
	`union all`
	`select parent.ID,parent.PID,parent.NAME from cte as child join b as parent on child.pid=parent.id`
	`where child.ID = a.ID`
	`)`
	`select NAME from cte limit 1`
	`) cName`
	`from`
	`(`
	`select id, name, count(*) as cnt`
	`from a group by id,name`
	`) a order by 1,2;`
		LOG: SQL can't be shipped, reason: With-Recursive recursive term correlated only is not shippable
		
		The correlation conditions of correlated subqueries are only in the recursion part, and the non-recursion part has no correlation condition.

7	`WITH recursive t_result AS (`
	`select * from(`
	`SELECT dm,sj_dm,name,1 as level`
	`FROM test_rec_part`
	`WHERE sj_dm < 10 order by dm limit 6 offset 2)`
	`UNION all`
	`SELECT t2.dm,t2.sj_dm,t2.name||' > '|| t1.name,t1.level+1`
	`FROM t_result t1`
	`JOIN test_rec_part t2 ON t2.sj_dm = t1.dm`
	`)`
	`SELECT * FROM t_result t;`
		LOG: SQL can't be shipped, reason: With-Recursive contains conflict distribution in none-recursive(Replicate) recursive(Hash)
		
		The **replicate** plan is used for **limit** in the non-recursion part but the **hash** plan is used in the recursion part, resulting in conflicts.

No.	Scenario	Cause of Not Supporting Pushdown
8	`with recursive cte as (select * from rec_tb4 where id<4 union all select h.id,h.parentID,h.name from (with recursive cte as (select * from rec_tb4 where id<4 union all select h.id,h.parentID,h.name from rec_tb4 h inner join cte c on h.id=c.parentID) SELECT id ,parentID,name from cte order by parentID) h inner join cte c on h.id=c.parentID) SELECT id ,parentID,name from cte order by parentID,1,2,3;`	LOG: SQL can't be shipped, reason: Recursive CTE references recursive CTE "cte" **recursive** of multiple-layers are nested. That is, a **recursive** is nested in the recursion part of another **recursive**.

Functions That Do Not Support Pushdown

This module describes the variability of functions. The function variability in GaussDB(DWS) is as follows:

- **IMMUTABLE**

  Indicates that the function always returns the same result if the parameter values are the same.
- **STABLE**

  Indicates 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 values, but that its result varies by SQL statements.
- **VOLATILE**

  Indicates that the function value can change even within a single table scan, so no optimizations can be made.

The volatility of a function can be obtained by querying its **provolatile** column in **pg_proc**. The value **i** indicates immutable, **s** indicates stable, and **v** indicates volatile. The valid values of the **proshippable** column in **pg_proc** are **t**, **f**, and **NULL**. This column and the **provolatile** column together describe whether a function is pushed down.

- If the **provolatile** of a function is **i**, the function can be pushed down regardless of the value of **proshippable**.
- If the **provolatile** of a function is **s** or **v**, the function can be pushed only if the value of **proshippable** is **t**.
- CTEs containing random are not pushed down, because pushdown may lead to incorrect results.

For a UDF, you can specify the values of **provolatile** and **proshippable** during its creation. For details, see CREATE FUNCTION.

In scenarios where a function does not support pushdown, perform one of the following as required:

- If it is a system function, replace it with a functionally equivalent one.
- If it is a UDF function, check whether its **provolatile** and **proshippable** are correctly defined.

Example: UDF

Define a user-defined function that generates fixed output for a certain input as the **immutable** type.

Use the TPCDS sales information as an example. You need to define a function to obtain the discount information.

```
CREATE FUNCTION func_percent_2 (NUMERIC, NUMERIC) RETURNS NUMERIC
AS 'SELECT $1 / $2 WHERE $2 > 0.01'
LANGUAGE SQL
VOLATILE;
```

Run the following statement:

```
SELECT func_percent_2(ss_sales_price, ss_list_price)
FROM store_sales;
```

The execution plan is as follows:

```
 Data Node Scan on store_sales "_REMOTE_TABLE_QUERY_"
   Output: func_percent_2(store_sales.ss_sales_price, store_sales.ss_list_price)
   Remote query: SELECT ss_sales_price, ss_list_price FROM ONLY store_sales WHERE true
(3 rows)
```

func_percent_2 is not pushed down, and **ss_sales_price** and **ss_list_price** are executed on a CN. In this case, a large amount of resources on the CN is consumed, and the performance deteriorates as a result.

In this example, the function returns certain output when certain input is entered. Therefore, we can modify the function to the following one:

```
CREATE FUNCTION func_percent_1 (NUMERIC, NUMERIC) RETURNS NUMERIC
AS 'SELECT $1 / $2 WHERE $2 > 0.01'
LANGUAGE SQL
IMMUTABLE;
```

Run the following statement:

```
SELECT func_percent_1(ss_sales_price, ss_list_price)
FROM store_sales;
```

The execution plan is as follows:

```
 Data Node Scan on "__REMOTE_FQS_QUERY__"  (cost=0.00..0.00 rows=0 width=0)
   Output: (func_percent_1(store_sales.ss_sales_price, store_sales.ss_list_price))
   Node/s: All datanodes
   Remote query: SELECT public.func_percent_1(ss_sales_price, ss_list_price) AS func_percent_1 FROM public.store_sales
(4 rows)
```

func_percent_1 is pushed down to DNs for quicker execution. (In TPCDS 1000X, where three CNs and 18 DNs are used, the query efficiency is improved by over 100 times).

Example 2: Pushing Down the Sorting Operation

For details, see **Case: Pushing Down Sort Operations to DNs**.

13.4.7.3 Optimizing Subqueries

What Is a Subquery

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 easy to understand. Therefore, subqueries are widely used.

In GaussDB(DWS), subqueries can also be called sublinks based on the location of subqueries in SQL statements.

- Subquery: corresponds to a scope 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 scope table and a sublink is an expression in the query parsing tree. A sublink can be found in constraint conditions and expressions. In GaussDB(DWS), sublinks can be classified into the following types:

  - exist_sublink: corresponding to the **EXIST** and **NOT EXIST** statements.
  - any_sublink: corresponding to the **OP ANY(SELECT...)** statement. **OP** can be the **IN**, **<**, **>**, or = operator.
  - all_sublink: corresponding to the **OP ALL(SELECT...)** statement. **OP** can be the **IN**, **<**, **>**, or = operator.
  - rowcompare_sublink: corresponding to the **RECORD OP (SELECT...)** statement.
  - expr_sublink: corresponding to the **(SELECT** with a single target list item**)** statement.
  - array_sublink: corresponding to the **ARRAY(SELECT...)** statement.
  - cte_sublink: corresponding to the **WITH(...)** statement.

  The sublinks commonly used in OLAP and HTAP are exist_sublink and any_sublink. The sublinks are pulled up by the optimization engine of GaussDB(DWS). Because of the flexible use of subqueries in SQL statements, complex subqueries may affect query performance. Subqueries are classified into non-correlated subqueries and correlated subqueries.

  - **Non-correlated subquery**

    The execution of a subquery is independent from any attribute of outer queries. 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
    ---------------------------------------------------------------
    Streaming (type: GATHER)
    ```

```
   Node/s: All datanodes
   ->  Hash Right Semi Join
         Hash Cond: (t2.c2 = t1.c1)
         ->  Streaming(type: REDISTRIBUTE)
               Spawn on: All datanodes
               ->  Seq Scan on t2
                     Filter: (c2 = ANY ('{2,3,4}'::integer[]))
         ->  Hash
               ->  Seq Scan on t1
(10 rows)
```

– **Correlated subquery**

The execution of a subquery depends on some attributes of outer queries which are used as **AND** conditions of the subquery. In the following example, **t1.c1** in the **t2.c1 = t1.c1** condition is a dependent attribute. Such a subquery depends on outer queries and needs to be executed once for each outer query.

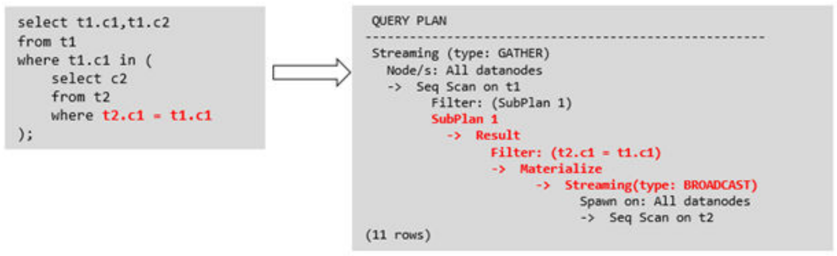
Example:

```
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
-----------------------------------------------------------------------
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
                                Filter: (c2 = ANY ('{2,3,4}'::integer[]))
(12 rows)
```

GaussDB(DWS) SubLink Optimization

A subquery is pulled up to join with tables in outer queries, preventing the subquery from being converted into the combination of a subplan and broadcast. You can run the **EXPLAIN** statement to check whether a subquery is converted into the combination of a subplan and broadcast.

Example:

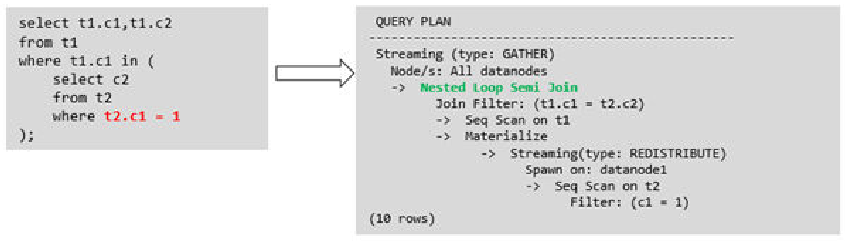


```
select t1.c1,t1.c2
from t1
where t1.c1 in (
    select c2
    from t2
    where t2.c1 = t1.c1
);
```

```
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)
```

- Sublink-release supported by GaussDB(DWS)

– 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.

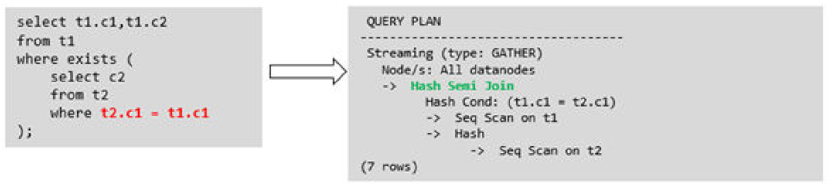


```
select t1.c1,t1.c2
from t1
where t1.c1 in (
    select c2
    from t2
    where t2.c1 = 1
);
```

```
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)
```

– 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)
```

– Pulling up an equivalent query containing aggregation 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 expression in the **WHERE** condition of the subquery must be table columns.
  - 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 aggregation 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 inner join.

  For example, the following subquery can 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 a **EXIST**-related sublink connected by **OR**,

for 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 procedure for promoting the OR clause of an EXIST-related subquery in an OR-ed join is as follows:

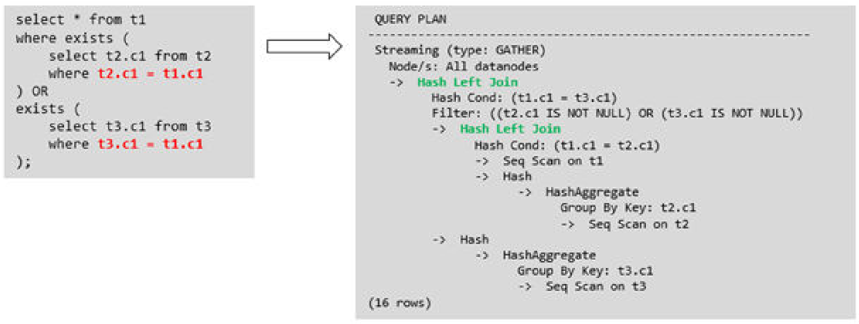
i. Extract **opExpr** from the **OR** clause in the **WHERE** condition. The value is **t1.a = (select avg(a) from t3 where t1.b = t3.b)**.

ii. The **opExpr** contains a subquery. If the subquery can be pulled up, the subquery is rewritten as **elect 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 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)
left join (select t4.c from t4 group by t4.c) where t3.b is not null or t4.c is not null;
```



```
select * from t1
where exists (
    select t2.c1 from t2
    where t2.c1 = t1.c1
) OR
exists (
    select t3.c1 from t3
    where t3.c1 = t1.c1
);
```

```
 QUERY PLAN
-----------------------------------------------------------------
 Streaming (type: GATHER)
   Node/s: All datanodes
   ->  Hash Left Join
         Hash Cond: (t1.c1 = t3.c1)
         Filter: ((t2.c1 IS NOT NULL) OR (t3.c1 IS NOT NULL))
         ->  Hash Left Join
               Hash Cond: (t1.c1 = t2.c1)
               ->  Seq Scan on t1
               ->  Hash
                     ->  HashAggregate
                           Group By Key: t2.c1
                           ->  Seq Scan on t2
         ->  Hash
               ->  HashAggregate
                     Group By Key: t3.c1
                     ->  Seq Scan on t3
(16 rows)
```

- Sublink-release not supported by GaussDB(DWS)

  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 a subplan 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 parent 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
-----------------------------------------------------
 Streaming (type: GATHER)
   Node/s: All datanodes
   ->  Seq Scan on t1
         Filter: (c2 > 10)
         SubPlan 1
           ->  Result
                 Filter: (t1.c1 = t2.c1)
                 ->  Materialize
                       ->  Streaming(type: BROADCAST)
                             Spawn on: All datanodes
                             ->  Seq Scan on t2
(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 a left outer join to join **t1** and **t2** so that the 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 only a single row and column scalar value
- CSSQ: an SSQ containing conditions

The preceding SQL statement can be changed into:

```
with ssq as
(
    select t2.c1, t2.c2 from t2
)
select ssq.c2, t1.c2
from t1 left join ssq on t1.c1 = ssq.c1
where t1.c2 > 10;
```

The execution plan after the change is as follows:

```
                QUERY PLAN
-------------------------------------------
 Streaming (type: GATHER)
   Node/s: All datanodes
   ->  Hash Right Join
         Hash Cond: (t2.c1 = t1.c1)
               ->  Seq Scan on t2
         ->  Hash
               ->  Seq Scan on t1
                     Filter: (c2 > 10)
(8 rows)
```

In the preceding example, the SSQ is pulled up to right join, preventing poor performance caused by the combination of a subplan and broadcast 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
-----------------------------------------------------------------
 Streaming (type: GATHER)
   Node/s: All datanodes
   ->  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
                       ->  Result
                             Filter: (t2.c1 = t1.c1)
                             ->  Materialize
                                   ->  Streaming(type: BROADCAST)
                                         Spawn on: All datanodes
                                         ->  Seq Scan on t2
(17 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 a left outer join to join **t1** and **t2** so that the 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. **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
-----------------------------------------------------
 Streaming (type: GATHER)
   Node/s: All datanodes
   ->  Sort
         Sort Key: (count(*)), t1.c1
         ->  Hash Join
               Hash Cond: (t1.c1 = t3.c1)
               ->  Hash Left Join
                     Hash Cond: (t1.c1 = t2.c1)
                     ->  Seq Scan on t1
                     ->  Hash
                           ->  HashAggregate
                                 Group By Key: t2.c1
                                 ->  Seq Scan on t2
               ->  Hash
                     ->  Seq Scan on t3
(15 rows)
```

– Pulling up nonequivalent subqueries

Example:

```
select t1.c1, t1.c2
from t1
where t1.c1 = (select agg() from t2.c2 > t1.c2);
```

Nonequivalent 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

- Currently, GaussDB(DWS) does not have an effective way to provide globally unique row IDs for tables and intermediate result sets. Therefore, the rewriting is difficult. It is recommended that this issue is avoided at the service layer or by using **t1.xc_node_id + t1.ctid** to associate row IDs. However, the high repetition rate of **xc_node_id** leads to low association efficiency, and **xc_node_id+ctid** cannot be used as the join condition of hash join.
- If the AGG type is **COUNT(*)**, **0** is used for data padding if **CASE-WHEN** is not matched. If the type is not **COUNT(*)**, **NULL** is used.
- CTE rewriting works better by using share scan.

More Optimization Examples

1. Change the base table to a replication table and create an index on the filter column.

```
create table master_table (a int);
create table sub_table(a int, b int);
select a from master_table group by a having a in (select a from sub_table);
```

In this example, a correlated subquery is contained. To improve the query performance, you can change **sub_table** to a replication table and create an index on the **a** column.

2. Modify the **SELECT** statement, change the subquery to a **JOIN** relationship between the primary table and the parent query, or modify the subquery to improve the query performance. Ensure that the subquery to be used is semantically correct.

```
explain (costs off)select * from master_table as t1 where t1.a in (select t2.a from sub_table as t2 where t1.a
= t2.b);
                        QUERY PLAN
-----------------------------------------------------------
Streaming (type: GATHER)
   Node/s: All datanodes
   ->  Seq Scan on master_table t1
         Filter: (SubPlan 1)
         SubPlan 1
           ->  Result
                 Filter: (t1.a = t2.b)
                 ->  Materialize
                       ->  Streaming(type: BROADCAST)
               Spawn on: All datanodes
                             ->  Seq Scan on sub_table t2
(11 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 master_table as t1 where exists (select t2.a from sub_table as t2 where t1.a
= t2.b and t1.a = t2.a);
                        QUERY PLAN
-------------------------------------------------
Streaming (type: GATHER)
   Node/s: All datanodes
   ->  Hash Semi Join
         Hash Cond: (t1.a = t2.b)
         ->  Seq Scan on master_table t1
         ->  Hash
         ->  Streaming(type: REDISTRIBUTE)
               Spawn on: All datanodes
                     ->  Seq Scan on sub_table t2
(9 rows)
```

In this way, the subplan is replaced by the semi-join between the two tables, greatly improving the execution efficiency.

13.4.7.4 Optimizing Statistics

What Is Statistic Optimization

GaussDB(DWS) 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

The query performance is often significantly impacted by the absence of statistics for tables or columns involved in the query.

The structure of the example table 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) distribute by hash(L_ORDERKEY);

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) distribute by hash(O_ORDERKEY);
```

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;
```

You can perform the following operations to check whether **ANALYZE** has been executed on the tables or columns involved in the query to collect statistics.

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,l_commitdate,l_orderkey,
   l_suppkey), public.orders(o_orderstatus,o_orderkey)) are not collected.
   HINT:Do analyze for them in order to generate optimized plan.
   ```
2. To determine if poor query performance was caused by a lack of statistics in certain tables or columns, check if the following information exists in the log file located in the **pg_log** directory.
   ```
   2017-06-14 17:28:30.336 CST 140644024579856 20971684 [BACKEND] LOG:Statistics in some tables
   or columns(public.lineitem(l_receiptdate, l_commitdate,l_orderkey,
   .l_suppkey), public.orders(o_orderstatus,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.
   ```

After confirming that **ANALYZE** has not been executed on the relevant tables or columns, you can execute **ANALYZE** on the tables or columns reported in the

WARNING or logs to resolve the issue of slow query performance due to a lack of statistics

Example 2: Setting cost_param to Optimize Query Performance

For details, see **Case: Configuring cost_param for Better Query Performance**.

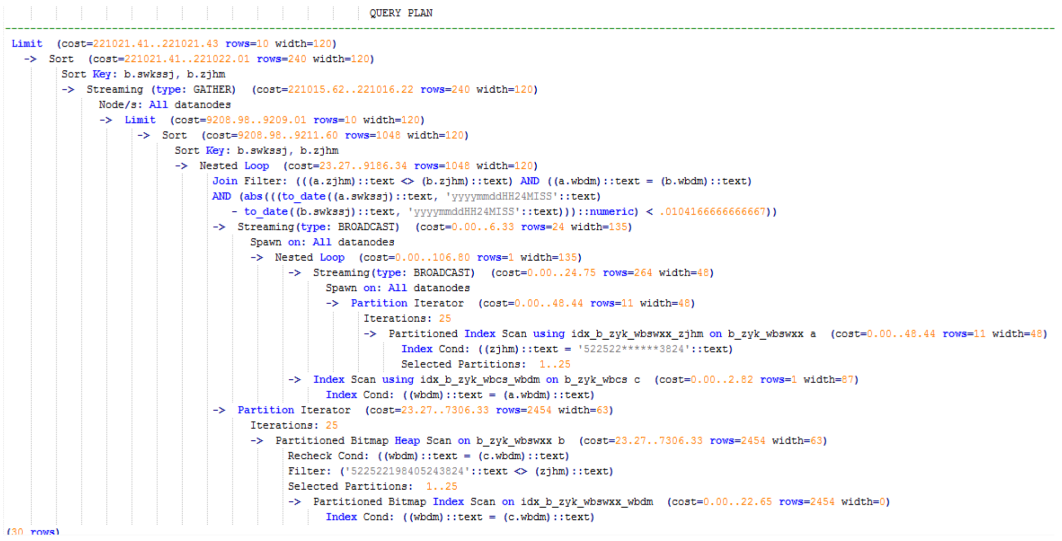
Example 3: Optimization is Not Accurate When Intermediate Results Exist in the Query Where JOIN Is Used for Multiple Tables

Symptom: Query the personnel who have checked in an Internet cafe within 15 minutes before and after the check-in of a specified person.

```
SELECT
C.WBM,
C.DZQH,
C.DZ,
B.ZJHM,
B.SWKSSJ,
B.XWSJ
FROM
b_zyk_wbswxx A,
b_zyk_wbswxx B,
b_zyk_wbcs C
WHERE
A.ZJHM = '522522******3824'
AND A.WBDM = B.WBDM
AND A.WBDM = C.WBDM
AND abs(to_date(A.SWKSSJ,'yyyymmddHH24MISS') - to_date(B.SWKSSJ,'yyyymmddHH24MISS')) <
INTERVAL '15 MINUTES'
ORDER BY
B.SWKSSJ,
B.ZJHM
limit 10 offset 0
;
```

Figure 13-6 shows the execution plan. This query takes about 12s.

Figure 13-6 Using an unlogged table (1)



```
                                                  QUERY PLAN
--------------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=221021.41..221021.43 rows=10 width=120)
   ->  Sort  (cost=221021.41..221022.01 rows=240 width=120)
         Sort Key: b.swkssj, b.zjhm
         ->  Streaming (type: GATHER)  (cost=221015.62..221016.22 rows=240 width=120)
               Node/s: All datanodes
               ->  Limit  (cost=9208.98..9209.01 rows=10 width=120)
                     ->  Sort  (cost=9208.98..9211.60 rows=1048 width=120)
                           Sort Key: b.swkssj, b.zjhm
                           ->  Nested Loop  (cost=23.27..9186.34 rows=1048 width=120)
                                 Join Filter: (((a.zjhm)::text <> (b.zjhm)::text) AND ((a.wbdm)::text = (b.wbdm)::text)
                                 AND (abs(((to_date((a.swkssj)::text, 'yyyymmddHH24MISS'::text)
                                    - to_date((b.swkssj)::text, 'yyyymmddHH24MISS'::text)))::numeric) < .0104166666666667))
                                 ->  Streaming(type: BROADCAST)  (cost=0.00..6.33 rows=24 width=135)
                                       Spawn on: All datanodes
                                       ->  Nested Loop  (cost=0.00..106.80 rows=1 width=135)
                                             ->  Streaming(type: BROADCAST)  (cost=0.00..24.75 rows=264 width=48)
                                                   Spawn on: All datanodes
                                                   ->  Partition Iterator  (cost=0.00..48.44 rows=11 width=48)
                                                         Iterations: 25
                                                         ->  Partitioned Index Scan using idx_b_zyk_wbswxx_zjhm on b_zyk_wbswxx a  (cost=0.00..48.44 rows=11 width=48)
                                                               Index Cond: ((zjhm)::text = '522522******3824'::text)
                                                               Selected Partitions:  1..25
                                             ->  Index Scan using idx_b_zyk_wbcs_wbdm on b_zyk_wbcs c  (cost=0.00..2.82 rows=1 width=87)
                                                   Index Cond: ((wbdm)::text = (a.wbdm)::text)
                                 ->  Partition Iterator  (cost=23.27..7306.33 rows=2454 width=63)
                                       Iterations: 25
                                       ->  Partitioned Bitmap Heap Scan on b_zyk_wbswxx b  (cost=23.27..7306.33 rows=2454 width=63)
                                             Recheck Cond: ((wbdm)::text = (c.wbdm)::text)
                                             Filter: ('522522198405243824'::text <> (zjhm)::text)
                                             Selected Partitions:  1..25
                                             ->  Partitioned Bitmap Index Scan on idx_b_zyk_wbswxx_wbdm  (cost=0.00..22.65 rows=2454 width=0)
                                                   Index Cond: ((wbdm)::text = (c.wbdm)::text)
(30 rows)
```

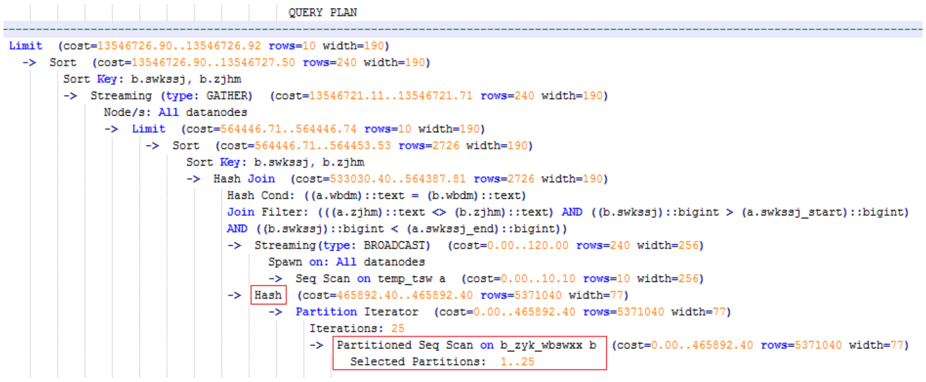
Optimization analysis:

1. In the execution plan, index scan is used for node scanning, the **Join Filter** calculation in the external **NEST LOOP IN** statement consumes most of the query time, and the calculation uses the string addition and subtraction, and unequal-value comparison.
2. Use an unlogged table to record the Internet access time of the specified person. The start time and end time are processed during data insertion, and this reduces subsequent addition and subtraction operations.

```
//Create a temporary unlogged table.
CREATE UNLOGGED TABLE temp_tsw
(
ZJHM         NVARCHAR2(18),
WBDM         NVARCHAR2(14),
SWKSSJ_START NVARCHAR2(14),
SWKSSJ_END   NVARCHAR2(14),
WBM          NVARCHAR2(70),
DZQH         NVARCHAR2(6),
DZ           NVARCHAR2(70),
IPDZ         NVARCHAR2(39)
)
;
//Insert the Internet access record of the specified person, and process the start time and end time.
INSERT INTO
temp_tsw
SELECT
A.ZJHM,
A.WBDM,
to_char((to_date(A.SWKSSJ,'yyyymmddHH24MISS') - INTERVAL '15
MINUTES'),'yyyymmddHH24MISS'),
to_char((to_date(A.SWKSSJ,'yyyymmddHH24MISS') + INTERVAL '15
MINUTES'),'yyyymmddHH24MISS'),
B.WBM,B.DZQH,B.DZ,B.IPDZ
FROM
b_zyk_wbswxx A,
b_zyk_wbcs B
WHERE
A.ZJHM='522522******3824' AND A.WBDM = B.WBDM
;

//Query the personnel who have check in an Internet cafe before and after 15 minutes of the check-in
of the specified person. Convert their ID card number format to int8 in comparison.
SELECT
A.WBM,
A.DZQH,
A.DZ,
A.IPDZ,
B.ZJHM,
B.XM,
to_date(B.SWKSSJ,'yyyymmddHH24MISS') as SWKSSJ,
to_date(B.XWSJ,'yyyymmddHH24MISS') as XWSJ,
B.SWZDH
FROM temp_tsw A,
b_zyk_wbswxx B
WHERE
A.ZJHM <> B.ZJHM
AND A.WBDM = B.WBDM
AND (B.SWKSSJ)::int8 > (A.swkssj_start)::int8
AND (B.SWKSSJ)::int8 < (A.swkssj_end)::int8
order by
B.SWKSSJ,
B.ZJHM
limit 10 offset 0
;
```

The query takes about 7s. **Figure 13-7** shows the execution plan.

Figure 13-7 Using an unlogged table (2)



```
                                        QUERY PLAN
----------------------------------------------------------------------------------------------------------
Limit  (cost=13546726.90..13546726.92 rows=10 width=190)
  ->  Sort  (cost=13546726.90..13546727.50 rows=240 width=190)
        Sort Key: b.swkssj, b.zjhm
        ->  Streaming (type: GATHER)  (cost=13546721.11..13546721.71 rows=240 width=190)
              Node/s: All datanodes
              ->  Limit  (cost=564446.71..564446.74 rows=10 width=190)
                    ->  Sort  (cost=564446.71..564453.53 rows=2726 width=190)
                          Sort Key: b.swkssj, b.zjhm
                          ->  Hash Join  (cost=533030.40..564387.81 rows=2726 width=190)
                                Hash Cond: ((a.wbdm)::text = (b.wbdm)::text)
                                Join Filter: (((a.zjhm)::text <> (b.zjhm)::text) AND ((b.swkssj)::bigint > (a.swkssj_start)::bigint)
                                AND ((b.swkssj)::bigint < (a.swkssj_end)::bigint))
                                ->  Streaming(type: BROADCAST)  (cost=0.00..120.00 rows=240 width=256)
                                      Spawn on: All datanodes
                                      ->  Seq Scan on temp_tsw a  (cost=0.00..10.10 rows=10 width=256)
                                ->  Hash  (cost=465892.40..465892.40 rows=5371040 width=77)
                                      ->  Partition Iterator  (cost=0.00..465892.40 rows=5371040 width=77)
                                            Iterations: 25
                                            ->  Partitioned Seq Scan on b_zyk_wbswxx b  (cost=0.00..465892.40 rows=5371040 width=77)
                                                  Selected Partitions:  1..25
```

3. In the previous plan, **Hash Join** has been executed, and a Hash table has been created for the large table **b_zyk_wbswxx**. The table contains large amounts of data, so the creation takes long time.

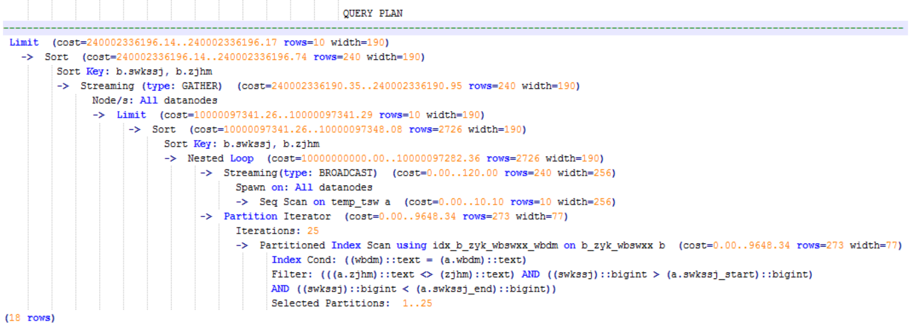
   temp_tsw contains only hundreds of records, and an equal-value connection is created between **temp_tsw** and **b_zyk_wbswxx** using wbdm (the Internet cafe code). Therefore, if **JOIN** is changed to **NEST LOOP JOIN**, index scan can be used for node scanning, and the performance will be boosted.

4. Execute the following statement to change **JOIN** to **NEST LOOP JOIN**.

   ```
   SET enable_hashjoin = off;
   ```

   Figure 13-8 shows the execution plan. The query takes about 3s.

Figure 13-8 Using an unlogged table (3)



```
                                                  QUERY PLAN
--------------------------------------------------------------------------------------------------------------------------
Limit  (cost=240002336196.14..240002336196.17 rows=10 width=190)
  ->  Sort  (cost=240002336196.14..240002336196.74 rows=240 width=190)
        Sort Key: b.swkssj, b.zjhm
        ->  Streaming (type: GATHER)  (cost=240002336190.35..240002336190.95 rows=240 width=190)
              Node/s: All datanodes
              ->  Limit  (cost=10000097341.26..10000097341.29 rows=10 width=190)
                    ->  Sort  (cost=10000097341.26..10000097348.08 rows=2726 width=190)
                          Sort Key: b.swkssj, b.zjhm
                          ->  Nested Loop  (cost=10000000000.00..10000097282.36 rows=2726 width=190)
                                ->  Streaming(type: BROADCAST)  (cost=0.00..120.00 rows=240 width=256)
                                      Spawn on: All datanodes
                                      ->  Seq Scan on temp_tsw a  (cost=0.00..10.10 rows=10 width=256)
                                ->  Partition Iterator  (cost=0.00..9648.34 rows=273 width=77)
                                      Iterations: 25
                                      ->  Partitioned Index Scan using idx_b_zyk_wbswxx_wbdm on b_zyk_wbswxx b  (cost=0.00..9648.34 rows=273 width=77)
                                            Index Cond: ((wbdm)::text = (a.wbdm)::text)
                                            Filter: (((a.zjhm)::text <> (zjhm)::text) AND ((swkssj)::bigint > (a.swkssj_start)::bigint)
                                            AND ((swkssj)::bigint < (a.swkssj_end)::bigint))
                                            Selected Partitions:  1..25
(18 rows)
```

5. Save the query result set in the unlogged table for paging display.

   If paging display needs to be achieved on the upper-layer application page, change the **offset** value to determine the result set on the target page. In this way, the previous query statement will be executed every time after a page turning operation, which causes long response latency.

   To resolve this problem, you are advised to use the unlogged table to save the result set.

   ```
   //Create an unlogged table to save the result set.
   CREATE UNLOGGED TABLE temp_result
   (
   WBM       NVARCHAR2(70),
   DZQH      NVARCHAR2(6),
   DZ        NVARCHAR2(70),
   IPDZ      NVARCHAR2(39),
   ZJHM      NVARCHAR2(18),
   XM        NVARCHAR2(30),
   SWKSSJ    date,
   XWSJ      date,
   ```

```
SWZDH    NVARCHAR2(32)
);

//Insert the result set to the unlogged table. The insertion takes about 3s.
INSERT INTO
temp_result
SELECT
A.WBM,
A.DZQH,
A.DZ,
A.IPDZ,
B.ZJHM,
B.XM,
to_date(B.SWKSSJ,'yyyymmddHH24MISS') as SWKSSJ,
to_date(B.XWSJ,'yyyymmddHH24MISS') as XWSJ,
B.SWZDH
FROM temp_tsw A,
b_zyk_wbswxx B
WHERE
A.ZJHM <> B.ZJHM
AND A.WBDM = B.WBDM
AND (B.SWKSSJ)::int8 > (A.swkssj_start)::int8
AND (B.SWKSSJ)::int8 < (A.swkssj_end)::int8
;

//Perform paging query on the result set. The paging query takes about 10 ms.
SELECT
*
FROM
temp_result
ORDER BY
SWKSSJ,
ZJHM
LIMIT 10 OFFSET 0;
```

CAUTION

Collecting global statistics using ANALYZE improves query performance.

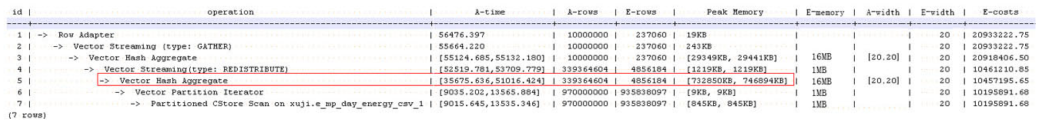
If a performance problem occurs, you can use plan hint to adjust the query plan to the previous one. For details, see **Hint-based Tuning**.

13.4.7.5 Optimizing Operators

What Is Operator Optimization

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/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)
```

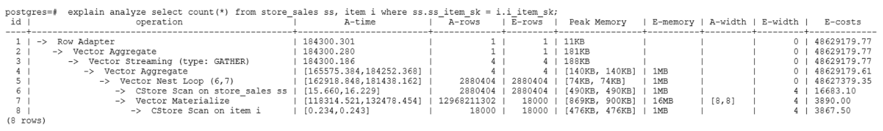
Operator Optimization Example

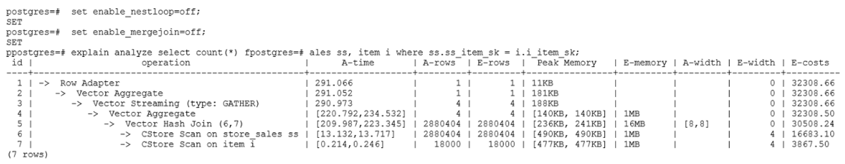
1. Scan the base table. For queries requiring large volume of data filtering, such as point queries or queries that need range scanning, a full table scan using SeqScan will take a long time. To facilitate scanning, you can create indexes on the condition column and select IndexScan for index scanning.

```
 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] |

 Predicate Information (identified by plan id)
-----------------------------------------------
   2 --Seq Scan on store_sales
         Filter: (ss_sold_date_sk = 2450944)
         Rows Removed by Filter: 4968936
 create index idx on store_sales_row(ss_sold_date_sk);
CREATE INDEX
 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] |
```

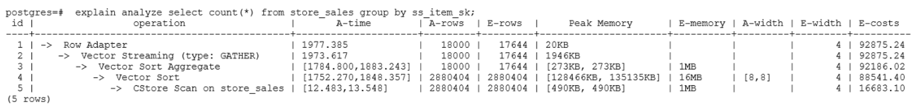
In this example, the full table scan filters much data and returns 3360 records. After an index has been created on the **ss_sold_date_sk** column, the scanning efficiency is significantly boosted from 3.6s to 13 ms by using **IndexScan**.

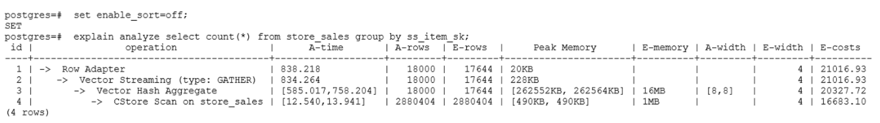
2: If NestLoop is used for joining tables with a large number of rows, the join may take a long time. In the following example, NestLoop takes 181s. If **enable_mergejoin=off** is set to disable merge join and **enable_nestloop=off** is set to disable NestLoop so that the optimizer selects hash join, the join takes more than 200 ms.


```
postgres=#  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)
```


```
postgres=#  set enable_nestloop=off;
SET
postgres=#  set enable_mergejoin=off;
SET
ppostgres=# 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**.


```
postgres=#  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)
```


```
postgres=#  set enable_sort=off;
SET
postgres=#  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)
```

13.4.7.6 Optimizing Data Skew

Data skew breaks the balance among nodes in the distributed MPP architecture. If the amount of data stored or processed by a node is much greater than that by other nodes, the following problems may occur:

- Storage skew severely limits the system capacity. The skew on a single node hinders system storage utilization.
- Computing skew severely affects performance. The data to be processed on the skew node is much more than that on other nodes, deteriorating overall system performance.
- Data skew severely affects the scalability of the MPP architecture. During storage or computing, data with the same values is often placed on the same node. Therefore, even if you add nodes after a data skew occurs, the skew data (data with the same values) is still placed on the node and affects the system capacity or performance bottleneck.

GaussDB(DWS) provides a complete solution for data skew, including storage and computing skew.

Data Skew in the Storage Layer

In the GaussDB(DWS) database, data is distributed and stored on each DN. You can improve the query efficiency by using distributed execution. However, if data skew occurs, bottlenecks exist on some DNs during distribution execution, affecting the query performance. This is because the distribution column is not properly selected. This can be solved by adjusting the distribution column.

For example:

```
explain performance select count(*) from inventory;
5 --CStore Scan on lmz.inventory
        dn_6001_6002 (actual time=0.444..83.127 rows=42000000 loops=1)
        dn_6003_6004 (actual time=0.512..63.554 rows=27000000 loops=1)
        dn_6005_6006 (actual time=0.722..99.033 rows=45000000 loops=1)
        dn_6007_6008 (actual time=0.529..100.379 rows=51000000 loops=1)
        dn_6009_6010 (actual time=0.382..71.341 rows=36000000 loops=1)
        dn_6011_6012 (actual time=0.547..100.274 rows=51000000 loops=1)
        dn_6013_6014 (actual time=0.596..118.289 rows=60000000 loops=1)
        dn_6015_6016 (actual time=1.057..132.346 rows=63000000 loops=1)
        dn_6017_6018 (actual time=0.940..110.310 rows=54000000 loops=1)
        dn_6019_6020 (actual time=0.231..41.198 rows=21000000 loops=1)
        dn_6021_6022 (actual time=0.927..114.538 rows=54000000 loops=1)
        dn_6023_6024 (actual time=0.637..118.385 rows=60000000 loops=1)
        dn_6025_6026 (actual time=0.288..32.240 rows=15000000 loops=1)
        dn_6027_6028 (actual time=0.566..118.096 rows=60000000 loops=1)
        dn_6029_6030 (actual time=0.423..82.913 rows=42000000 loops=1)
        dn_6031_6032 (actual time=0.395..78.103 rows=39000000 loops=1)
        dn_6033_6034 (actual time=0.376..51.052 rows=24000000 loops=1)
        dn_6035_6036 (actual time=0.569..79.463 rows=39000000 loops=1)
```

In the performance information, you can view the number of scan rows of each DN in the inventory table. The number of rows of each DN differs a lot, the biggest is 63000000 and the smallest value is 15000000. This value difference on the performance of data scan is acceptable, but if the join operator exists in the upper-layer, the impact on the performance cannot be ignored.

Generally, the data table is hash distributed on each DN; therefore, it is important to choose a proper distribution column. Run table_skewness() to view data skew of each DN in the inventory table. The query result is as follows:

```
select table_skewness('inventory');
              table_skewness
------------------------------------------
 ("dn_6015_6016        ",63000000,8.046%)
 ("dn_6013_6014        ",60000000,7.663%)
 ("dn_6023_6024        ",60000000,7.663%)
 ("dn_6027_6028        ",60000000,7.663%)
 ("dn_6017_6018        ",54000000,6.897%)
 ("dn_6021_6022        ",54000000,6.897%)
 ("dn_6007_6008        ",51000000,6.513%)
 ("dn_6011_6012        ",51000000,6.513%)
 ("dn_6005_6006        ",45000000,5.747%)
 ("dn_6001_6002        ",42000000,5.364%)
 ("dn_6029_6030        ",42000000,5.364%)
 ("dn_6031_6032        ",39000000,4.981%)
 ("dn_6035_6036        ",39000000,4.981%)
 ("dn_6009_6010        ",36000000,4.598%)
 ("dn_6003_6004        ",27000000,3.448%)
 ("dn_6033_6034        ",24000000,3.065%)
 ("dn_6019_6020        ",21000000,2.682%)
 ("dn_6025_6026        ",15000000,1.916%)
(18 rows)
```

The table definition indicates that the table uses the **inv_date_sk** column as the distribution column, which causes a data skew. Based on the data distribution of each column, change the distribution column to **inv_item_sk**. The skew status is as follows:

```
select table_skewness('inventory');
              table_skewness
------------------------------------------
 ("dn_6001_6002        ",43934200,5.611%)
 ("dn_6007_6008        ",43829420,5.598%)
 ("dn_6003_6004        ",43781960,5.592%)
 ("dn_6031_6032        ",43773880,5.591%)
 ("dn_6033_6034        ",43763280,5.589%)
 ("dn_6011_6012        ",43683600,5.579%)
 ("dn_6013_6014        ",43551660,5.562%)
 ("dn_6027_6028        ",43546340,5.561%)
 ("dn_6009_6010        ",43508700,5.557%)
 ("dn_6023_6024        ",43484540,5.554%)
 ("dn_6019_6020        ",43466800,5.551%)
 ("dn_6021_6022        ",43458500,5.550%)
 ("dn_6017_6018        ",43448040,5.549%)
 ("dn_6015_6016        ",43247700,5.523%)
 ("dn_6005_6006        ",43200240,5.517%)
 ("dn_6029_6030        ",43181360,5.515%)
 ("dn_6025_6026        ",43179700,5.515%)
 ("dn_6035_6036        ",42960080,5.487%)
(18 rows)
```

Data skew is solved.

In addition to the **table_skewness()** view, you can use the **table_distribution** function and the PGXC_GET_TABLE_SKEWNESS view to efficiently query the data skew status of each table.

Data Skew in the Computing Layer

Even if data is balanced across nodes after you change the distribution key of a table, data skew may still occur during a query. If data skew occurs in the result set of an operator on a DN, skew will also occur during the computing that involves the operator. Generally, this is caused by data redistribution during the execution.

During a query, JOIN keys and GROUP BY keys are not used as distribution columns. Data is redistributed among DNs based on the hash values of data on the keys. The redistribution is implemented using the Redistribute operator in an execution plan. Data skew in redistribution columns can lead to data skew during system operation. After the redistribution, some nodes will have much more data, process more data, and will have much lower performance than others.

In the following example, the **s** and **t** tables are joined, and **s.x** and **t.x** columns in the join condition are not their distribution keys. Table data is redistributed using the **REDISTRIBUTE** operator. Data skew occurs in the **s.x** column and not in the **t.x** column. The result set of the **Streaming** operator (**id** being **6**) on datanode2 has data three times that of other DNs and causes a skew.

```
select * from skew s,test t where s.x = t.x order by s.a limit 1;
 id |                    operation                     |       A-time
----+---------------------------------------------------+-----------------------
  1 | ->  Limit                                          | 52622.382
  2 |    ->  Streaming (type: GATHER)                    | 52622.374
  3 |       ->  Limit                                    | [30138.494,52598.994]
  4 |          ->  Sort                                  | [30138.486,52598.986]
  5 |             ->  Hash Join (6,8)                    | [30127.013,41483.275]
  6 |                ->  Streaming(type: REDISTRIBUTE)   | [11365.110,22024.845]
  7 |                   ->  Seq Scan on public.skew s    | [2019.168,2175.369]
  8 |                ->  Hash                            | [2460.108,2499.850]
  9 |                   ->  Streaming(type: REDISTRIBUTE) | [1056.214,1121.887]
 10 |                      ->  Seq Scan on public.test t  | [310.848,325.569]

6 --Streaming(type: REDISTRIBUTE)
        datanode1 (rows=5050368)
        datanode2 (rows=15276032)
        datanode3 (rows=5174272)
        datanode4 (rows=5219328)
```

Computing skew is more difficult to detect than storage skew. To solve computing skew, GaussDB provides the Runtime Load Balance Technology (RLBT) solution, controlled by the **skew_option** parameter. The RLBT solution addresses how to detect and solve data skew.

1. Detect data skew.

   The solution first checks whether skew data exists in redistribution columns used for computing. RLBT can detect data skew based on statistics, specified hints, or rules.

   – Detection based on statistics

     Run the **ANALYZE** statement to collect statistics on tables. The optimizer will automatically identify skew data on redistribution keys based on the statistics and generate optimization plans for queries having potential skew. When the redistribution key has multiple columns, statistics information can be used for identification only when all columns belong to the same base table.

     The statistics information can only provide the skew of the base table. If a column in the base table is skewed, or other columns have filtering conditions, or after the join of other tables, we cannot determine whether the skewed data still exists on the skewed column. If **skew_option** is **normal**, it indicates that the skew data still exists, and the base tables will be optimized to solve skew. If **skew_option** is **lazy**, it indicates that no more skew data exists and the optimization will stop.

   – Detection based on specified hints

The intermediate results of complex queries are difficult to estimate based on statistics. In this case, you can specify hints to provide the skew information, based on which the optimizer optimizes queries. For details about the syntax of hints, see **Skew Hints**.

– Detection based on rules

In a business intelligence (BI) system, a large number of SQL statements having outer joins (including left joins, right joins, and full joins) are generated, and many NULL values will be generated in empty columns that have no match for outer joins. If JOIN or GROUP BY operations are performed on the columns, data skew will occur. RLBT can automatically identify this scenario and generate an optimization plan for NULL value skew.

2. Solve computing skew.

Join and **Aggregate** operators are optimized to solve skew.

– **Join** optimization

Skew and non-skew data is separately processed. Details are as follows:

a. When redistribution is required on both sides of a join:

Use **PART_REDISTRIBUTE_PART_ROUNDROBIN** on the side with skew. Specifically, perform round-robin on skew data and redistribution on non-skew data.

Use **PART_REDISTRIBUTE_PART_BROADCAST** on the side with no skew. Specifically, perform broadcast on skew data and redistribution on non-skew data.

b. When redistribution is required on only one side of a join:

Use **PART_REDISTRIBUTE_PART_ROUNDROBIN** on the side where redistribution is required.

Use **PART_LOCAL_PART_BROADCAST** on the side where redistribution is not required. Specifically, perform broadcast on skew data and retain other data locally.

c. When a table has **NULL** values padded:

Use **PART_REDISTRIBUTE_PART_LOCAL** on the table. Specifically, retain the **NULL** values locally and perform redistribution on other data.

In the example query, the **s.x** column contains skewed data and its value is **0**. The optimizer identifies the skew data in statistics and generates the following optimization plan:

```
id |                              operation                              |        A-time
----+-----------------------------------------------------------------------------+-----------------------
  1 | ->  Limit                                                          | 23642.049
  2 |   ->  Streaming (type: GATHER)                                     | 23642.041
  3 |      ->  Limit                                                     | [23310.768,23618.021]
  4 |         ->  Sort                                                   | [23310.761,23618.012]
  5 |            ->  Hash Join (6,8)                                     | [20898.341,21115.272]
  6 |               ->  Streaming(type: PART REDISTRIBUTE PART ROUNDROBIN)   |
[7125.834,7472.111]
  7 |                  ->  Seq Scan on public.skew s                     | [1837.079,1911.025]
  8 |               ->  Hash                                             | [2612.484,2640.572]
  9 |                  ->  Streaming(type: PART REDISTRIBUTE PART BROADCAST) | [1193.548,1297.894]
 10 |                     ->  Seq Scan on public.test t                  | [314.343,328.707]

  5 --Vector Hash Join (6,8)
        Hash Cond: s.x = t.x
        Skew Join Optimized by Statistic
  6 --Streaming(type: PART REDISTRIBUTE PART ROUNDROBIN)
        datanode1 (rows=7635968)
        datanode2 (rows=7517184)
        datanode3 (rows=7748608)
        datanode4 (rows=7818240)
```

In the preceding execution plan, **Skew Join Optimized by Statistic** indicates that this is an optimized plan used for handling data skew. The **Statistic** keyword indicates that the plan optimization is based on statistics; **Hint** indicates that the optimization is based on hints; **Rule** indicates that the optimization is based on rules. In this plan, skew and non-skew data is separately processed. Non-skew data in the **s** table is redistributed based on its hash values, and skew data (whose value is **0**) is evenly distributed on all nodes in round-robin mode. In this way, data skew is solved.

To ensure result correctness, the **t** table also needs to be processed. In the **t** table, the data whose value is **0** (skew value in the **s.x** table) is broadcast and other data is redistributed based on its hash values.

In this way, data skew in JOIN operations is solved. The above result shows that the output of the **Streaming** operator (**id** being **6**) is balanced and the end-to-end performance of the query is doubled.

If the stream operator type in the execution plan is **HYBRID**, the stream mode varies depending on the skew data. The following plan is an example:

```
EXPLAIN (nodes OFF, costs OFF) SELECT COUNT(*) FROM skew_scol s, skew_scol1 s1 WHERE s.b =
s1.c;
QUERY PLAN
----------------------------------------------------------------------------------------------------------------------
------------------------------------------
id |                                                operation
----
+---------------------------------------------------------------------------------------------------------------------
--------------------------------------
1 | ->  Aggregate
2 |    ->  Streaming (type: GATHER)
3 |       ->  Aggregate
4 |          ->  Hash Join (5,7)
5 |             ->  Streaming(type: HYBRID)
6 |                ->  Seq Scan on skew_scol s
7 |             ->  Hash
8 |                ->  Streaming(type: HYBRID)
9 |                   ->  Seq Scan on skew_scol1 s1

Predicate Information (identified by plan id)
----------------------------------------------------------------------------------------------------------------------
--------------------
4 --Hash Join (5,7)
Hash Cond: (s.b = s1.c)
Skew Join Optimized by Statistic
5 --Streaming(type: HYBRID)
Skew Filter: (b = 1)
Skew Filter: (b = 0)
8 --Streaming(type: HYBRID)
Skew Filter: (c = 0)
Skew Filter: (c = 1)
```

Data 1 has skew in the **skew_scol** table. Perform **ROUNDROBIN** on skew data and **REDISTRIBUTE** on non-skew data.

Data 0 is the side with no skew in the **skew_scol** table. Perform **BROADCAST** on skew data and **REDISTRIBUTE** on non-skew data.

As shown in the preceding figure, the two stream types are **PART REDISTRIBUTE PART ROUNDROBIN** and **PART REDISTRIBUTE PART BROADCAST**. In this example, the stream type is **HYBRID**.

– **Aggregate** optimization

For aggregation, data on each DN is deduplicated based on the **GROUP BY** key and then redistributed. After the deduplication on DNs, the global occurrences of each value will not be greater than the number of DNs. Therefore, no serious data skew will occur. Take the following query as an example:

```
select c1, c2, c3, c4, c5, c6, c7, c8, c9, count(*) from t group by c1, c2, c3, c4, c5, c6, c7, c8, c9 limit 10;
```

The command output is as follows:

```
 id |                 operation                  |         A-time          |  A-rows
----+--------------------------------------------+-------------------------+----------
  1 | ->  Streaming (type: GATHER)               | 130621.783              |       12
  2 |    ->  GroupAggregate                      | [85499.711,130432.341] |       12
  3 |       ->  Sort                             | [85499.509,103145.632] | 36679237
  4 |          ->  Streaming(type: REDISTRIBUTE) | [25668.897,85499.050]  | 36679237
  5 |             ->  Seq Scan on public.t       | [9835.069,10416.388]   | 36679237

   4 --Streaming(type: REDISTRIBUTE)
         datanode1 (rows=36678837)
         datanode2 (rows=100)
         datanode3 (rows=100)
         datanode4 (rows=200)
```

A large amount of skew data exists. As a result, after data is redistributed based on its **GROUP BY** key, the data volume of datanode1 is hundreds of thousands of times that of others. After optimization, a GROUP BY operation is performed on the DN to deduplicate data. After redistribution, no data skew occurs.

```
 id |                 operation                  |        A-time
----+--------------------------------------------+-----------------------
  1 | ->  Streaming (type: GATHER)               | 10961.337
  2 |    ->  HashAggregate                       | [10953.014,10953.705]
  3 |       ->  HashAggregate                    | [10952.957,10953.632]
  4 |          ->  Streaming(type: REDISTRIBUTE) | [10952.859,10953.502]
  5 |             ->  HashAggregate              | [10084.280,10947.139]
  6 |                ->  Seq Scan on public.t    | [4757.031,5201.168]

 Predicate Information (identified by plan id)
-----------------------------------------------
   3 --HashAggregate
         Skew Agg Optimized by Statistic

   4 --Streaming(type: REDISTRIBUTE)
         datanode1 (rows=17)
         datanode2 (rows=8)
         datanode3 (rows=8)
         datanode4 (rows=14)
```

Applicable scope

– **Join** operator

  - **nest loop**, **merge join**, and **hash join** can be optimized.
  - If skew data is on the left to the join, **inner join**, **left join**, **semi join**, and **anti join** are supported. If skew data is on the right to the join, **inner join**, **right join**, **right semi join**, and **right anti join** are supported.
  - For an optimization plan generated based on statistics, the optimizer checks whether it is optimal by estimating its cost. Optimization plans based on hints or rules are forcibly generated.

– **Aggregate** operator

  - **array_agg**, **string_agg**, and **subplan in agg qual** cannot be optimized.
  - A plan generated based on statistics is affected by its cost, the **plan_mode_seed** parameter, and the **best_agg_plan** parameter. A plan generated based on hints or rules are not affected by them.

13.4.7.7 SQL Statement Rewriting Rules

Based on the database SQL execution mechanism and a large number of practices, summarize finds that: using rules of a certain SQL statement, on the basis of the so that the correct test result, which can improve the SQL execution efficiency. You can comply with these rules to greatly improve service query efficiency.

- Replacing **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.

- **Adding NOT NULL to the join column**

  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.

- Converting **NOT IN** to **NOT EXISTS**

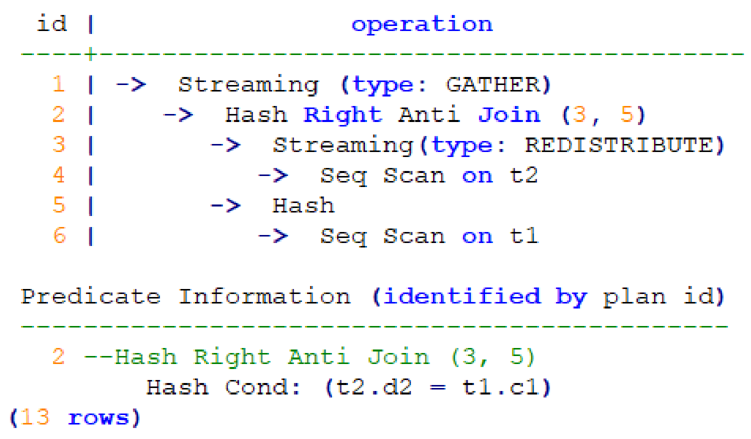
  nestloop anti join must be used to implement **NOT IN**, and **Hash anti join** is required for **NOT EXISTS**. If no **NULL** value exists in the **JOIN** column, **NOT IN** is equivalent to **NOT EXISTS**. Therefore, if you are sure that no **NULL** value exists, you can convert **NOT IN** to **NOT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

  As shown in the following figure, 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:

  Figure 13-9 NOT EXISTS execution plan



```
 id |                operation
----+-----------------------------------------
  1 | ->  Streaming (type: GATHER)
  2 |    ->  Hash Right Anti Join (3, 5)
  3 |       ->  Streaming(type: REDISTRIBUTE)
  4 |          ->  Seq Scan on t2
  5 |       ->  Hash
  6 |          ->  Seq Scan on t1

Predicate Information (identified by plan id)
---------------------------------------------
  2 --Hash Right Anti Join (3, 5)
        Hash Cond: (t2.d2 = t1.c1)
(13 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(DWS) performance greatly deteriorates if a large number of functions are called. In this case, you can modify 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 task 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.
  - **BROADCAST** subqueries are performed on large tables in multi-DN environment.

13.4.7.8 Tuning Optimizer Parameters

This section introduces optimizer parameters that affect the performance of SQL statements used in GaussDB(DWS). For details about the parameter configuration method, see **Configuring GUC Parameters**.

Table 13-13 CN parameters

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** To temporarily change the value of this parameter in the current database connection (that is, the current session), run 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), run the following SQL statements: `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_fast_query_shipping=on	Specifies whether the optimizer uses a distribution framework. If the value is **on**, the execution plan is generated on both CNs and DNs. If the value is **off**, the distribution framework is used, that is, the execution plan is generated on the CNs and then sent to DNs for execution. **NOTE** To temporarily change the value of this parameter in the current database connection (that is, the current session), run the following SQL statement: `SET enable_fast_query_shipping to off;`
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.

Parameter/ Reference Value	Description
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.
enable_seqscan=on	Specifies whether the optimizer uses bitmap scanning. It is impossible to suppress sequential scans entirely, but setting this variable to **off** allows the optimizer to preferentially 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.
enable_broadcast=on	Specifies whether enable the optimizer's use of data broadcast. In data broadcast, a large amount of data is transferred on the network. When the number of transmission nodes (stream) is large and the estimation is inaccurate, set this parameter to **off** and check whether the performance is improved.
rewrite_rule	Specifies whether the optimizer enables a specific rewriting rule.

13.4.8 Hint-based Tuning

13.4.8.1 Plan Hint Optimization

In plan hints, you can specify a join order, join, stream, and scan operations, the number of rows in a result, and redistribution skew information to tune an execution plan, improving query performance.

Function

The hint syntax must follow immediately after a **SELECT** keyword and is written in the following format:

```
/*+ <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.

For 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.

Supported Hints

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
- Stream operation hints
- Scan operation hints, supporting only **tablescan**, **indexscan**, and **indexonlyscan**
- Sublink name hints
- Skew hints, supporting only the skew in the redistribution involving Join or HashAgg
- Hint used for **Agg** distribution columns Only clusters of 8.1.3.100 and later versions support this function.
- Configuration parameter hints, supporting the parameters described in **Configuration Parameter Hints**

Precautions

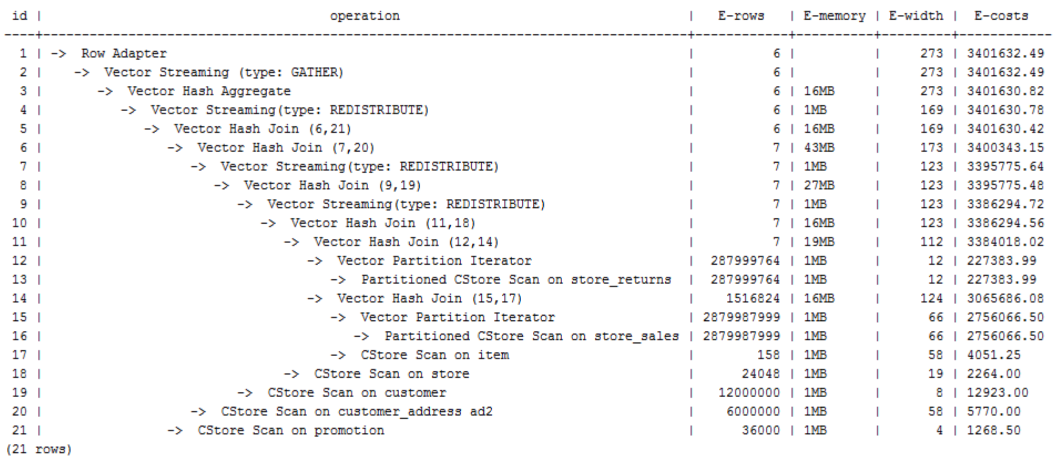
- **Sort**, **Setop**, and **Subplan** hints are not supported.
- Hints do not support SMP or Node Groups.
- Hints cannot be used for the target table of the **INSERT** statement.

Examples

The following is the original plan and is used for comparing with the optimized ones:

```
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
;
```



```
 id |                                   operation                                        |   E-rows   | E-memory | E-width |  E-costs
----+------------------------------------------------------------------------------------+------------+----------+---------+------------
  1 | ->  Row Adapter                                                                    |          6 |          |     273 | 3401632.49
  2 |    ->  Vector Streaming (type: GATHER)                                             |          6 |          |     273 | 3401632.49
  3 |       ->  Vector Hash Aggregate                                                    |          6 | 16MB     |     273 | 3401630.82
  4 |          ->  Vector Streaming(type: REDISTRIBUTE)                                  |          6 | 1MB      |     169 | 3401630.78
  5 |             ->  Vector Hash Join (6,21)                                            |          6 | 16MB     |     169 | 3401630.42
  6 |                ->  Vector Hash Join (7,20)                                         |          7 | 43MB     |     173 | 3400343.15
  7 |                   ->  Vector Streaming(type: REDISTRIBUTE)                         |          7 | 1MB      |     123 | 3395775.64
  8 |                      ->  Vector Hash Join (9,19)                                   |          7 | 27MB     |     123 | 3395775.48
  9 |                         ->  Vector Streaming(type: REDISTRIBUTE)                   |          7 | 1MB      |     123 | 3386294.72
 10 |                            ->  Vector Hash Join (11,18)                            |          7 | 16MB     |     123 | 3386294.56
 11 |                               ->  Vector Hash Join (12,14)                         |          7 | 19MB     |     112 | 3384018.02
 12 |                                  ->  Vector Partition Iterator                     |  287999764 | 1MB      |      12 | 227383.99
 13 |                                     ->  Partitioned CStore Scan on store_returns   |  287999764 | 1MB      |      12 | 227383.99
 14 |                                  ->  Vector Hash Join (15,17)                      |    1516824 | 16MB     |     124 | 3065686.08
 15 |                                     ->  Vector Partition Iterator                  | 2879987999 | 1MB      |      66 | 2756066.50
 16 |                                        ->  Partitioned CStore Scan on store_sales  | 2879987999 | 1MB      |      66 | 2756066.50
 17 |                                     ->  CStore Scan on item                        |        158 | 1MB      |      58 | 4051.25
 18 |                               ->  CStore Scan on store                             |      24048 | 1MB      |      19 | 2264.00
 19 |                         ->  CStore Scan on customer                                |   12000000 | 1MB      |       8 | 12923.00
 20 |                   ->  CStore Scan on customer_address ad2                          |    6000000 | 1MB      |      58 | 5770.00
 21 |                ->  CStore Scan on promotion                                        |      36000 | 1MB      |       4 | 1268.50
(21 rows)
```

13.4.8.2 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.

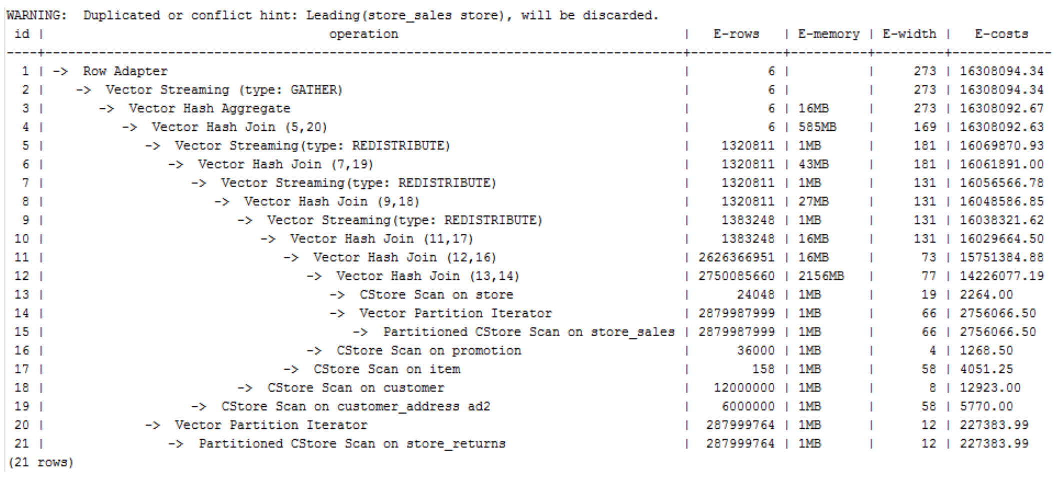
leading((t1 (t2 t3) t4 t5)) leading((t3 t2)): First, **t2** and **t3** are joined and **t2** is used as the inner table. Then, the result is joined with **t1**, and **(t2 t3)** is used as the inner table. Finally, the result is joined with **t4** and then **t5**, and the table on the right in each join is used as the inner table.

Examples

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.
 id |                                 operation                                    |   E-rows   | E-memory | E-width |   E-costs
----+------------------------------------------------------------------------------+------------+----------+---------+-------------
  1 | ->  Row Adapter                                                              |          6 |          |     273 | 16308094.34
  2 |   ->  Vector Streaming (type: GATHER)                                        |          6 |          |     273 | 16308094.34
  3 |     ->  Vector Hash Aggregate                                                |          6 | 16MB     |     273 | 16308092.67
  4 |       ->  Vector Hash Join (5,20)                                            |          6 | 585MB    |     169 | 16308092.63
  5 |         ->  Vector Streaming(type: REDISTRIBUTE)                             |    1320811 | 1MB      |     181 | 16069870.93
  6 |           ->  Vector Hash Join (7,19)                                        |    1320811 | 43MB     |     181 | 16061891.00
  7 |             ->  Vector Streaming(type: REDISTRIBUTE)                         |    1320811 | 1MB      |     131 | 16056566.78
  8 |               ->  Vector Hash Join (9,18)                                    |    1320811 | 27MB     |     131 | 16048586.85
  9 |                 ->  Vector Streaming(type: REDISTRIBUTE)                     |    1383248 | 1MB      |     131 | 16038321.62
 10 |                   ->  Vector Hash Join (11,17)                               |    1383248 | 16MB     |     131 | 16029664.50
 11 |                     ->  Vector Hash Join (12,16)                             | 2626366951 | 16MB     |      73 | 15751384.88
 12 |                       ->  Vector Hash Join (13,14)                           | 2750085660 | 2156MB   |      77 | 14226077.19
 13 |                         ->  CStore Scan on store                             |      24048 | 1MB      |      19 | 2264.00
 14 |                         ->  Vector Partition Iterator                        | 2879987999 | 1MB      |      66 | 2756066.50
 15 |                           ->  Partitioned CStore Scan on store_sales         | 2879987999 | 1MB      |      66 | 2756066.50
 16 |                       ->  CStore Scan on promotion                           |      36000 | 1MB      |       4 | 1268.50
 17 |                     ->  CStore Scan on item                                  |        158 | 1MB      |      58 | 4051.25
 18 |                 ->  CStore Scan on customer                                  |   12000000 | 1MB      |       8 | 12923.00
 19 |             ->  CStore Scan on customer_address ad2                          |    6000000 | 1MB      |      58 | 5770.00
 20 |         ->  Vector Partition Iterator                                        |  287999764 | 1MB      |      12 | 227383.99
 21 |           ->  Partitioned CStore Scan on store_returns                       |  287999764 | 1MB      |      12 | 227383.99
(21 rows)
```

For details about the warning at the top of the plan, see **Hint Errors, Conflicts, and Other Warnings**.

13.4.8.3 Join Operation Hints

Function

Specifies the join method. It 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:

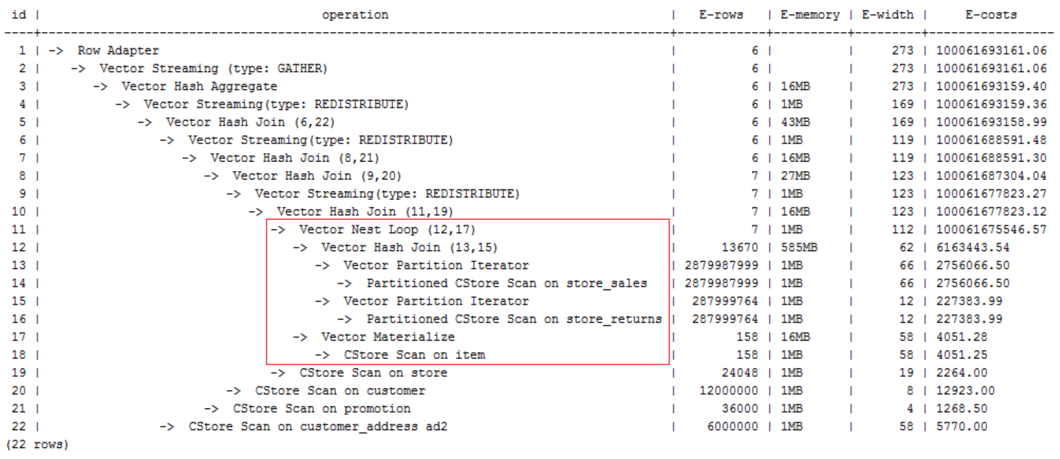
no nestloop(t1 t2 t3): **nestloop** is not used for joining **t1**, **t2**, and **t3**. The three tables may be joined in either of the two ways: Join **t2** and **t3**, and then **t1**; join **t1** and **t2**, and then **t3**. This hint takes effect only for the last join. If necessary, you can hint other joins. For example, you can add **no nestloop(t2 t3)** to join **t2** and **t3** first and to forbid the use of **nestloop**.

Examples

Hint the query plan in **Examples** 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:



```
 id |                                   operation                                    |   E-rows   | E-memory | E-width |     E-costs
----+--------------------------------------------------------------------------------+------------+----------+---------+-----------------
  1 | ->  Row Adapter                                                                |          6 |          |     273 | 100061693161.06
  2 |   ->  Vector Streaming (type: GATHER)                                          |          6 |          |     273 | 100061693161.06
  3 |     ->  Vector Hash Aggregate                                                  |          6 | 16MB     |     273 | 100061693159.40
  4 |       ->  Vector Streaming(type: REDISTRIBUTE)                                 |          6 | 1MB      |     169 | 100061693159.36
  5 |         ->  Vector Hash Join (6,22)                                            |          6 | 43MB     |     169 | 100061693158.99
  6 |           ->  Vector Streaming(type: REDISTRIBUTE)                             |          6 | 1MB      |     119 | 100061688591.48
  7 |             ->  Vector Hash Join (8,21)                                        |          6 | 16MB     |     119 | 100061688591.30
  8 |               ->  Vector Hash Join (9,20)                                      |          7 | 27MB     |     123 | 100061687304.04
  9 |                 ->  Vector Streaming(type: REDISTRIBUTE)                       |          7 | 1MB      |     123 | 100061677823.27
 10 |                   ->  Vector Hash Join (11,19)                                 |          7 | 16MB     |     123 | 100061677823.12
 11 |                     ->  Vector Nest Loop (12,17)                               |          7 | 1MB      |     112 | 100061675546.57
 12 |                       ->  Vector Hash Join (13,15)                             |      13670 | 585MB    |      62 | 6163443.54
 13 |                         ->  Vector Partition Iterator                          | 2879987999 | 1MB      |      66 | 2756066.50
 14 |                           ->  Partitioned CStore Scan on store_sales           | 2879987999 | 1MB      |      66 | 2756066.50
 15 |                         ->  Vector Partition Iterator                          |  287999764 | 1MB      |      12 | 227383.99
 16 |                           ->  Partitioned CStore Scan on store_returns         |  287999764 | 1MB      |      12 | 227383.99
 17 |                       ->  Vector Materialize                                   |        158 | 16MB     |      58 | 4051.28
 18 |                         ->  CStore Scan on item                                |        158 | 1MB      |      58 | 4051.25
 19 |                     ->  CStore Scan on store                                   |      24048 | 1MB      |      19 | 2264.00
 20 |                 ->  CStore Scan on customer                                    |   12000000 | 1MB      |       8 | 12923.00
 21 |               ->  CStore Scan on promotion                                     |      36000 | 1MB      |       4 | 1268.50
 22 |           ->  CStore Scan on customer_address ad2                              |    6000000 | 1MB      |      58 | 5770.00
(22 rows)
```

13.4.8.4 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.

For 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

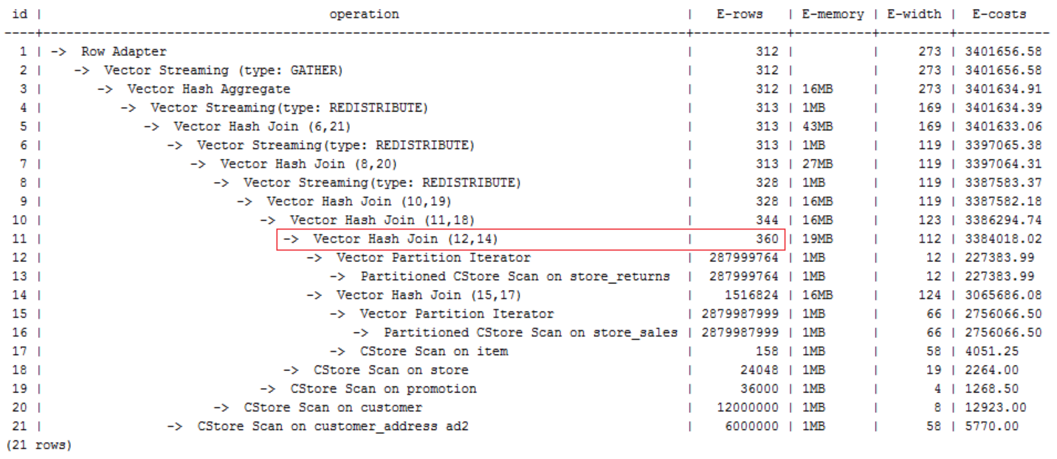
- The hint using ***** for two tables is recommended, because this hint will take effect for a join as long as the two tables appear on both sides of this join. For example, if the hint is **rows(t1 t2 * 3)**, the join result of **(t1 t3 t4)** and **(t2 t5 t6)** will be multiplied by 3 because **t1** 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:



```
 id |                          operation                           |   E-rows   | E-memory | E-width |  E-costs
----+--------------------------------------------------------------+------------+----------+---------+------------
  1 | ->  Row Adapter                                              |        312 |          |     273 | 3401656.58
  2 |   ->  Vector Streaming (type: GATHER)                        |        312 |          |     273 | 3401656.58
  3 |     ->  Vector Hash Aggregate                                |        312 | 16MB     |     273 | 3401634.91
  4 |       ->  Vector Streaming(type: REDISTRIBUTE)               |        313 | 1MB      |     169 | 3401634.39
  5 |         ->  Vector Hash Join (6,21)                          |        313 | 43MB     |     169 | 3401633.06
  6 |           ->  Vector Streaming(type: REDISTRIBUTE)           |        313 | 1MB      |     119 | 3397065.38
  7 |             ->  Vector Hash Join (8,20)                      |        313 | 27MB     |     119 | 3397064.31
  8 |               ->  Vector Streaming(type: REDISTRIBUTE)       |        328 | 1MB      |     119 | 3387583.37
  9 |                 ->  Vector Hash Join (10,19)                 |        328 | 16MB     |     119 | 3387582.18
 10 |                   ->  Vector Hash Join (11,18)               |        344 | 16MB     |     123 | 3386294.74
 11 |                     ->  Vector Hash Join (12,14)             |        360 | 19MB     |     112 | 3384018.02
 12 |                       ->  Vector Partition Iterator          |  287999764 | 1MB      |      12 | 227383.99
 13 |                         ->  Partitioned CStore Scan on store_returns |  287999764 | 1MB |      12 | 227383.99
 14 |                       ->  Vector Hash Join (15,17)           |    1516824 | 16MB     |     124 | 3065686.08
 15 |                         ->  Vector Partition Iterator        | 2879987999 | 1MB      |      66 | 2756066.50
 16 |                           ->  Partitioned CStore Scan on store_sales | 2879987999 | 1MB |      66 | 2756066.50
 17 |                         ->  CStore Scan on item              |        158 | 1MB      |      58 | 4051.25
 18 |                     ->  CStore Scan on store                 |      24048 | 1MB      |      19 | 2264.00
 19 |                   ->  CStore Scan on promotion               |      36000 | 1MB      |       4 | 1268.50
 20 |               ->  CStore Scan on customer                    |   12000000 | 1MB      |       8 | 12923.00
 21 |           ->  CStore Scan on customer_address ad2            |    6000000 | 1MB      |      58 | 5770.00
(21 rows)
```

The estimation value after the hint in row 11 is **360**, and the original value is rounded off to 7.

13.4.8.5 Stream Operation Hints

Function

Specifies the stream method, which can be broadcast, redistribute, or specifying the distribution key for **Agg** redistribution.

NOTE

Specifies the hint for the distribution column during the Agg process.. This parameter is supported only by clusters of version 8.1.3.100 or later.

Syntax

```
[no] broadcast | redistribute(table_list) | redistribute ((*) (columns))
```

Parameter Description

- **no** indicates that the hinted stream method is not used. When the hint is specified for the distribution columns in the **Agg** redistribution, **no** is invalid.
- *table_list* specifies the tables to be joined. For details, see **Parameter Description**.
- When hints are specified for distribution columns, the asterisk (*) is fixed and the table name cannot be specified.
- **columns** specifies one or more columns in the **GROUP BY** clause. When there are no **GROUP BY** clauses, it can specify the columns in the **DISTINCT** clause.'

NOTE

- The specified distribution column must be represented by the column number in **GROUP BY** or **DISTINCT**. The column name cannot be specified.
- For a multi-layer query, you can specify the distribution column hint at each layer. The hint takes effect only at the corresponding layer.
- If the optimizer finds that redistribution is not required after estimation, the specified distribution column is invalid.

Tips

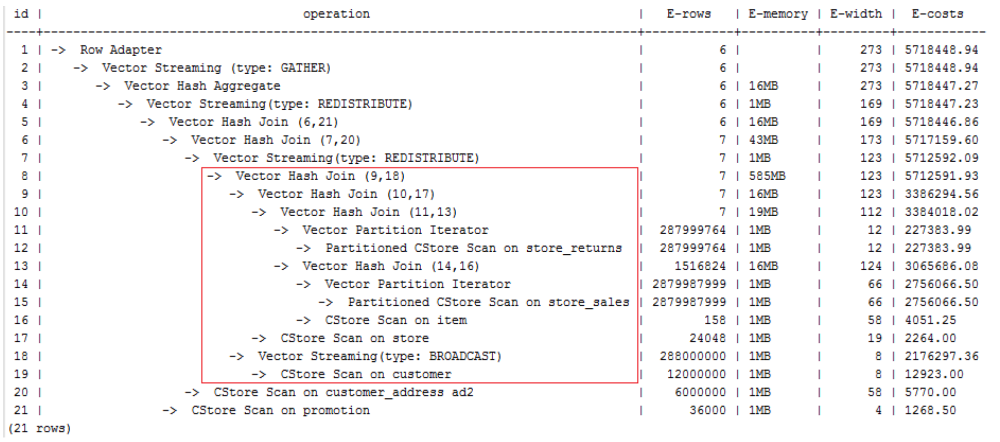
- Generally, the optimizer selects a group of non-skew distribution keys for data redistribution based on statistics. If the default distribution keys have data skew, you can manually specify the distribution columns to avoid data skew.
- When selecting a distribution key, select a group of columns with high distinct values as the distribution key based on data distribution features. In this way, data can be evenly distributed to each DN after redistribution.
- After writing hints, you can run **explain verbose** to print the execution plan and check whether the specified distribution key is valid. If the specified distribution key is invalid, a warning is displayed.

Example

- Hint the query plan in **Examples** as follows:

  ```
  explain
  select /*+ no redistribute(store_sales store_returns item store) leading(((store_sales store_returns item store) customer)) */ i_product_name product_name ...
  ```

  In the original plan, the join result of **store_sales**, **store_returns**, **item**, and **store** is redistributed before it is joined with **customer**. After the hinting, the redistribution is disabled and the join order is retained. The optimized plan is as follows:

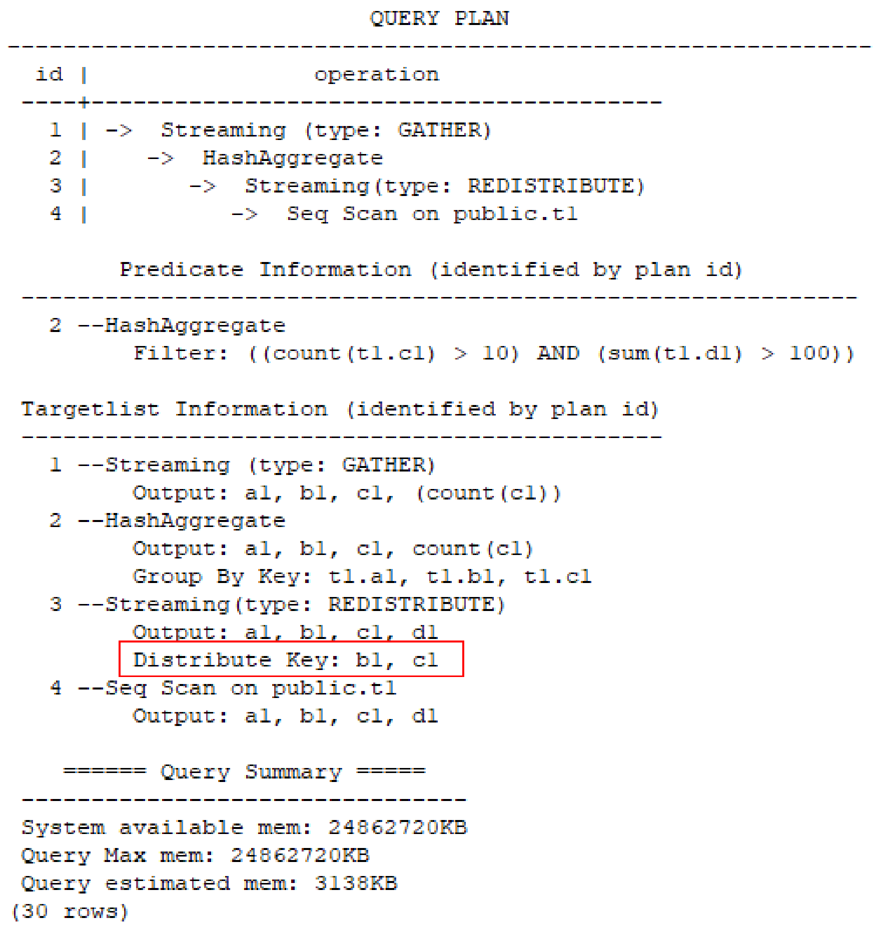
  

  ```
   id |                          operation                                    |   E-rows   | E-memory | E-width |  E-costs
  ----+-----------------------------------------------------------------------+------------+----------+---------+------------
    1 | ->  Row Adapter                                                       |          6 |          |     273 | 5718448.94
    2 |   ->  Vector Streaming (type: GATHER)                                 |          6 |          |     273 | 5718448.94
    3 |     ->  Vector Hash Aggregate                                         |          6 | 16MB     |     273 | 5718447.27
    4 |       ->  Vector Streaming(type: REDISTRIBUTE)                        |          6 | 1MB      |     169 | 5718447.23
    5 |         ->  Vector Hash Join (6,21)                                   |          6 | 16MB     |     169 | 5718446.86
    6 |           ->  Vector Hash Join (7,20)                                 |          7 | 43MB     |     173 | 5717159.60
    7 |             ->  Vector Streaming(type: REDISTRIBUTE)                  |          7 | 1MB      |     123 | 5712592.09
    8 |               ->  Vector Hash Join (9,18)                             |          7 | 585MB    |     123 | 5712591.93
    9 |                 ->  Vector Hash Join (10,17)                          |          7 | 16MB     |     123 | 3386294.56
   10 |                   ->  Vector Hash Join (11,13)                        |          7 | 19MB     |     112 | 3384018.02
   11 |                     ->  Vector Partition Iterator                     |  287999764 | 1MB      |      12 | 227383.99
   12 |                       ->  Partitioned CStore Scan on store_returns    |  287999764 | 1MB      |      12 | 227383.99
   13 |                     ->  Vector Hash Join (14,16)                      |    1516824 | 16MB     |     124 | 3065686.08
   14 |                       ->  Vector Partition Iterator                   | 2879987999 | 1MB      |      66 | 2756066.50
   15 |                         ->  Partitioned CStore Scan on store_sales    | 2879987999 | 1MB      |      66 | 2756066.50
   16 |                       ->  CStore Scan on item                         |        158 | 1MB      |      58 | 4051.25
   17 |                   ->  CStore Scan on store                            |      24048 | 1MB      |      19 | 2264.00
   18 |                 ->  Vector Streaming(type: BROADCAST)                 |  288000000 | 1MB      |       8 | 2176297.36
   19 |                   ->  CStore Scan on customer                         |   12000000 | 1MB      |       8 | 12923.00
   20 |             ->  CStore Scan on customer_address ad2                   |    6000000 | 1MB      |      58 | 5770.00
   21 |           ->  CStore Scan on promotion                                |      36000 | 1MB      |       4 | 1268.50
  (21 rows)
  ```

- Specifies the distribution columns for Agg redistribution.

  ```
  explain (verbose on, costs off, nodes off)
  select /*+ redistribute ((*) (2 3)) */ a1, b1, c1, count(c1)  from t1 group by a1, b1, c1 having count(c1) > 10 and sum(d1) > 100
  ```

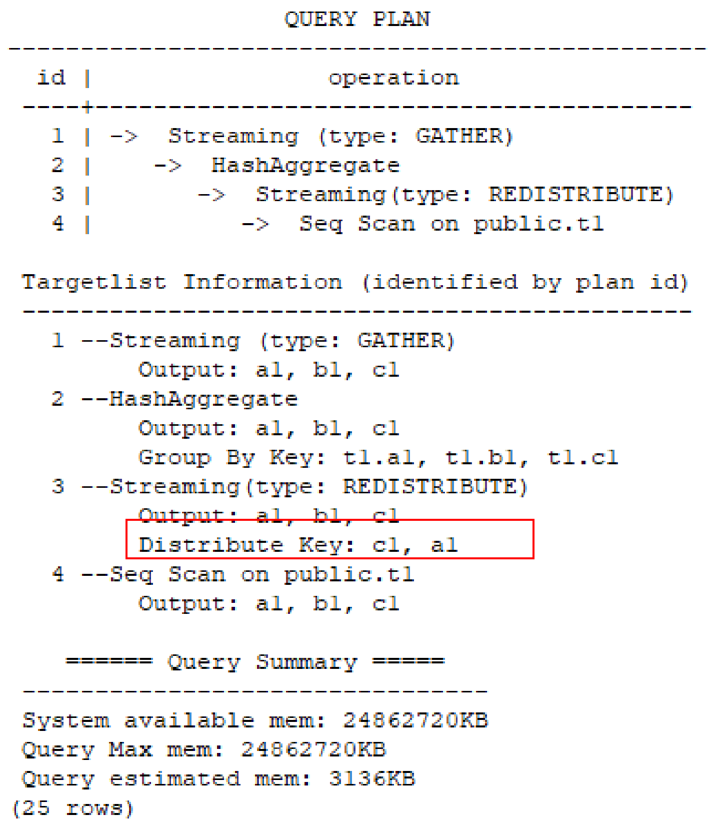
  In the following example, the last two columns of the specified **GROUP BY** columns are used as distribution keys.



```
                         QUERY PLAN
--------------------------------------------------------------
 id |                 operation
----+-------------------------------------------
  1 | ->  Streaming (type: GATHER)
  2 |    ->  HashAggregate
  3 |       ->  Streaming(type: REDISTRIBUTE)
  4 |          ->  Seq Scan on public.t1

       Predicate Information (identified by plan id)
------------------------------------------------------------
  2 --HashAggregate
        Filter: ((count(t1.c1) > 10) AND (sum(t1.d1) > 100))

 Targetlist Information (identified by plan id)
----------------------------------------------
  1 --Streaming (type: GATHER)
        Output: a1, b1, c1, (count(c1))
  2 --HashAggregate
        Output: a1, b1, c1, count(c1)
        Group By Key: t1.a1, t1.b1, t1.c1
  3 --Streaming(type: REDISTRIBUTE)
        Output: a1, b1, c1, d1
        Distribute Key: b1, c1
  4 --Seq Scan on public.t1
        Output: a1, b1, c1, d1

    ====== Query Summary =====
--------------------------------
 System available mem: 24862720KB
 Query Max mem: 24862720KB
 Query estimated mem: 3138KB
(30 rows)
```

- If the statement does not contain the **GROUP BY** clause, specify the distinct column as the distribution columns.

```
explain (verbose on, costs off, nodes off)
select /*+ redistribute ((*) (3 1)) */ distinct a1, b1, c1 from t1;
```



```
                         QUERY PLAN
-----------------------------------------------
 id |                  operation
----+------------------------------------------
  1 | ->  Streaming (type: GATHER)
  2 |    ->  HashAggregate
  3 |       ->  Streaming(type: REDISTRIBUTE)
  4 |          ->  Seq Scan on public.t1

 Targetlist Information (identified by plan id)
 ----------------------------------------------
  1 --Streaming (type: GATHER)
        Output: a1, b1, c1
  2 --HashAggregate
        Output: a1, b1, c1
        Group By Key: t1.a1, t1.b1, t1.c1
  3 --Streaming(type: REDISTRIBUTE)
        Output: a1, b1, c1
        Distribute Key: c1, a1
  4 --Seq Scan on public.t1
        Output: a1, b1, c1

    ====== Query Summary =====
 --------------------------------
 System available mem: 24862720KB
 Query Max mem: 24862720KB
 Query estimated mem: 3136KB
(25 rows)
```

13.4.8.6 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** indicates that the specified hint will not be used for a join.
- *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* indicates 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, HDFS foreign tables, OBS tables, and subquery tables. HDFS tables include primary tables and delta tables. The delta tables are invisible to users. Therefore, scan operation hints are used only for primary tables.

Example

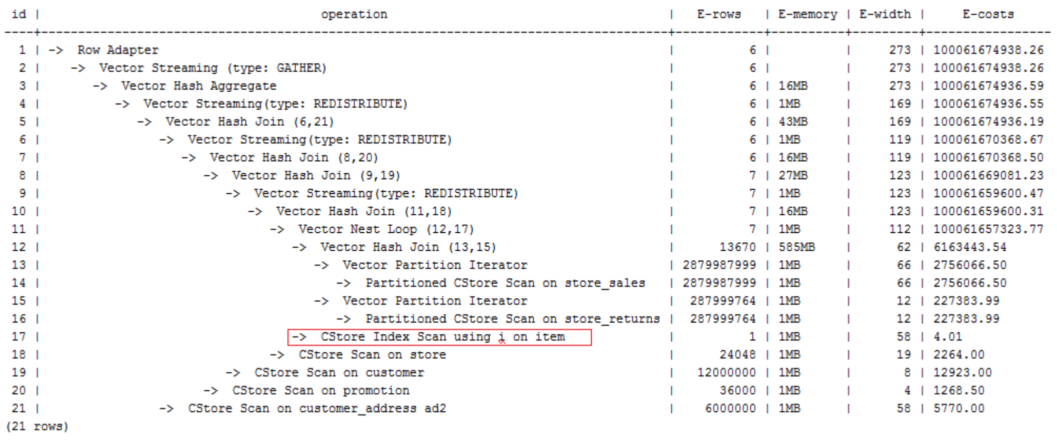
To specify an index-based hint for a scan, create an index named **i** on the **i_item_sk** column of the **item** table.

```
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:



```
 id |                                   operation                                   |   E-rows   | E-memory | E-width |     E-costs
----+-------------------------------------------------------------------------------+------------+----------+---------+-----------------
  1 | ->  Row Adapter                                                               |          6 |          |     273 | 100061674938.26
  2 |   ->  Vector Streaming (type: GATHER)                                         |          6 |          |     273 | 100061674938.26
  3 |     ->  Vector Hash Aggregate                                                 |          6 | 16MB     |     273 | 100061674936.59
  4 |       ->  Vector Streaming(type: REDISTRIBUTE)                                |          6 | 1MB      |     169 | 100061674936.55
  5 |         ->  Vector Hash Join (6,21)                                           |          6 | 43MB     |     169 | 100061674936.19
  6 |           ->  Vector Streaming(type: REDISTRIBUTE)                            |          6 | 1MB      |     119 | 100061670368.67
  7 |             ->  Vector Hash Join (8,20)                                       |          6 | 16MB     |     119 | 100061670368.50
  8 |               ->  Vector Hash Join (9,19)                                     |          7 | 27MB     |     123 | 100061669081.23
  9 |                 ->  Vector Streaming(type: REDISTRIBUTE)                      |          7 | 1MB      |     123 | 100061659600.47
 10 |                   ->  Vector Hash Join (11,18)                                |          7 | 16MB     |     123 | 100061659600.31
 11 |                     ->  Vector Nest Loop (12,17)                              |          7 | 1MB      |     112 | 100061657323.77
 12 |                       ->  Vector Hash Join (13,15)                            |      13670 | 585MB    |      62 | 6163443.54
 13 |                         ->  Vector Partition Iterator                         | 2879987999 | 1MB      |      66 | 2756066.50
 14 |                           ->  Partitioned CStore Scan on store_sales          | 2879987999 | 1MB      |      66 | 2756066.50
 15 |                         ->  Vector Partition Iterator                         |  287999764 | 1MB      |      12 | 227383.99
 16 |                           ->  Partitioned CStore Scan on store_returns        |  287999764 | 1MB      |      12 | 227383.99
 17 |                       ->  CStore Index Scan using i on item                   |          1 | 1MB      |      58 | 4.01
 18 |                     ->  CStore Scan on store                                  |      24048 | 1MB      |      19 | 2264.00
 19 |                 ->  CStore Scan on customer                                   |   12000000 | 1MB      |       8 | 12923.00
 20 |               ->  CStore Scan on promotion                                    |      36000 | 1MB      |       4 | 1268.50
 21 |           ->  CStore Scan on customer_address ad2                             |    6000000 | 1MB      |      58 | 5770.00
(21 rows)
```

13.4.8.7 Sublink Name Hints

Function

These hints specify the name of a sublink block.

Syntax

```
blockname (table)
```

Parameter Description

- *table* indicates the name you have specified for a sublink block.

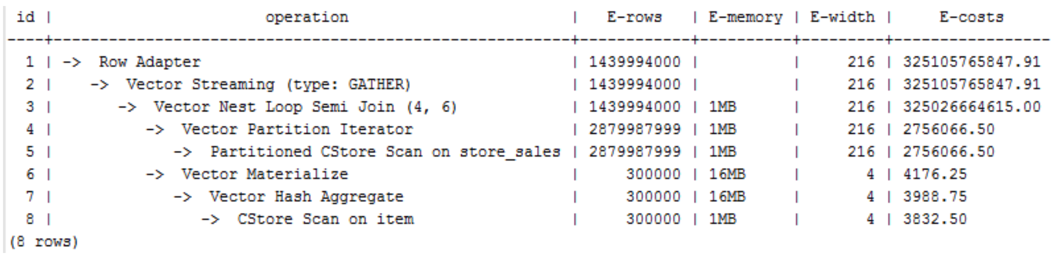
NOTE

- This hint is used by an outer query only when a sublink is 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.

Examples

```
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:



```
 id |                         operation                         |   E-rows   | E-memory | E-width |     E-costs
----+-----------------------------------------------------------+------------+----------+---------+-----------------
  1 | ->  Row Adapter                                           | 1439994000 |          |     216 | 325105765847.91
  2 |    ->  Vector Streaming (type: GATHER)                    | 1439994000 |          |     216 | 325105765847.91
  3 |       ->  Vector Nest Loop Semi Join (4, 6)               | 1439994000 | 1MB      |     216 | 325026664615.00
  4 |          ->  Vector Partition Iterator                    | 2879987999 | 1MB      |     216 | 2756066.50
  5 |             ->  Partitioned CStore Scan on store_sales    | 2879987999 | 1MB      |     216 | 2756066.50
  6 |          ->  Vector Materialize                           |     300000 | 16MB     |       4 | 4176.25
  7 |             ->  Vector Hash Aggregate                     |     300000 | 16MB     |       4 | 3988.75
  8 |                ->  CStore Scan on item                    |     300000 | 1MB      |       4 | 3832.50
(8 rows)
```

13.4.8.8 Skew Hints

Function

Theses hints specify redistribution keys containing skew data and skew values, and are used to optimize redistribution involving Join or HashAgg.

Syntax

- Specify single-table skew.
  ```
  skew(table (column) [(value)])
  ```
- Specify intermediate result skew.
  ```
  skew((join_rel) (column) [(value)])
  ```

Parameter Description

- **table** specifies the table where skew occurs.
- **join_rel** specifies two or more joined tables. For example, **(t1 t2)** indicates that the result of joining **t1** and **t2** tables contains skew data.
- **column** specifies one or more columns where skew occurs.
- **value** specifies one or more skew values.

NOTE

- Skew hints are used only if redistribution is required and the specified skew information matches the redistribution information.
- Skew hints are controlled by the GUC parameter . If the parameter is disabled, skew hints cannot be used for solving skew.
- Currently, skew hints support only the table relationships of the ordinary table and subquery types. Hints can be specified for base tables, subqueries, and **WITH ... AS** clauses. Unlike other hints, a subquery can be used in skew hints regardless of whether it is pulled up.
- Use an alias (if any) to specify a table where data skew occurs.
- You can use a name or an alias to specify a skew column as long as it is not ambiguous. The columns in skew hints cannot be expressions. If data skew occurs in the redistribution that uses an expression as a redistribution key, set the redistribution key as a new column and specify the column in skew hints.
- The number of skew values must be an integer multiple of the number of columns. Skew values must be grouped based on the column sequence, with each group containing a maximum of 10 values. You can specify duplicate values to group skew columns having different number of skew values. For example, the **c1** and **c2** columns of the **t1** table contains skew data. The skew value of the **c1** column is **a1**, and the skew values of the **c2** column are **b1** and **b2**. In this case, the skew hint is **skew(t1 (c1 c2) ((a1 b1)(a1 b2)))**. **(a1 b1)** is a value group, where **NULL** is allowed as a skew value. Each hint can contain a maximum of 10 groups and the number of groups should be an integer multiple of the number of columns.
- In the redistribution optimization of Join, a skew value must be specified for skew hints. The skew value can be left empty for HashAgg.
- If multiple tables, columns, or values are specified, separate items of the same type with spaces.
- The type of skew values cannot be forcibly converted in hints. To specify a string, enclose it with single quotation marks (' ').

Example:

- Specify single-table skew.

  Each skew hint describes the skew information of one table relationship. To describe the skews of multiple table relationships in a query, specify multiple skew hints.

  Skew hints have the following formats:

  – One skew value in one column: **skew(t (c1) (v1))**

    Description: The **v1** value in the **c1** column of the **t** table relationship causes skew in query execution.

  – Multiple skew values in one column: **skew(t (c1) (v1 v2 v3 ...))**

    Description: Values including **v1, v2**, and **v3** in the **c1** column of the **t** table relationship cause skew in query execution.

  – Multiple columns, each having one skew value: **skew(t (c1 c2) (v1 v2))**

    Description: The **v1** value in the **c1** column and the **v2** value in the **c2** column of the **t** table relationship cause skew in query execution.

  – Multiple columns, each having multiple skew values: **skew(t (c1 c2) ((v1 v2) (v3 v4) (v5 v6) ...))**

    Description: Values including **v1, v3**, and **v5** in the **c1** column and values including **v2, v4**, and **v6** in the **c2** column of the **t** table relationship cause skew in query execution.

NOTICE

In the last format, parentheses for skew value groups can be omitted, for example, **skew(t (c1 c2) (v1 v2 v3 v4 v5 v6 ...))**. In a skew hint, either use parentheses for all skew value groups or for none of them.

Otherwise, a syntax error will be generated. For example, **skew(t (c1 c2) (v1 v2 v3 v4 (v5 v6) ...))** will generate an error.

- Specify intermediate result skew.

  If data skew does not occur in base tables but in an intermediate result during query execution, specify skew hints of the intermediate result to solve the skew. The format is **skew((t1 t2) (c1) (v1))**.

  Description: Data skew occurs after the table relationships **t1** and **t2** are joined. The **c1** column of the **t1** table contains skew data and its skew value is **v1**.

  c1 can exist only in a table relationship of **join_rel**. If there is another column having the same name, use aliases to avoid ambiguity.

Suggestion

- For a multi-level query, write the hint on the layer where data skew occurs.
- For a listed subquery, you can specify the subquery name in a hint. If you know data skew occurs on which base table, directly specify the table.
- Aliases are preferred when you specify a table or column in a hint.

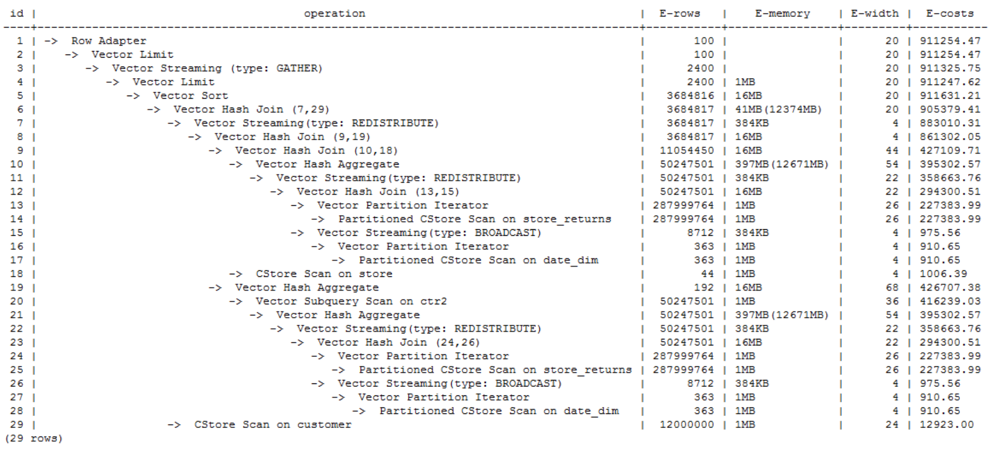
Examples

Specify single-table skew.

- Specify hints in the original query.

  For example, the original query is as follows:

```
explain
with customer_total_return as
(select sr_customer_sk as ctr_customer_sk
,sr_store_sk as ctr_store_sk
,sum(SR_FEE) as ctr_total_return
from store_returns
,date_dim
where sr_returned_date_sk = d_date_sk
and d_year =2000
group by sr_customer_sk
,sr_store_sk)
 select  c_customer_id
from customer_total_return ctr1
,store
,customer
where ctr1.ctr_total_return > (select avg(ctr_total_return)*1.2
from customer_total_return ctr2
where ctr1.ctr_store_sk = ctr2.ctr_store_sk)
and s_store_sk = ctr1.ctr_store_sk
and s_state = 'NM'
and ctr1.ctr_customer_sk = c_customer_sk
order by c_customer_id
limit 100;
```

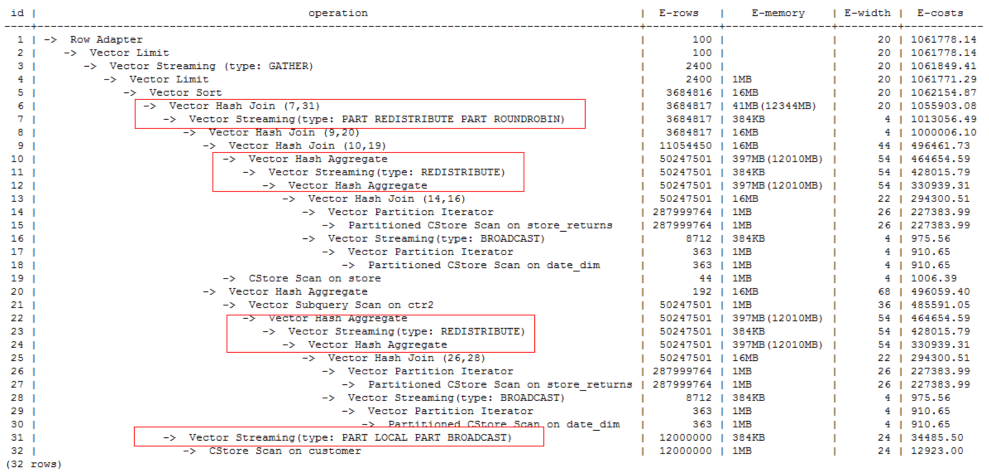


```
 id |                                       operation                                        |  E-rows   |    E-memory     | E-width |  E-costs
----+----------------------------------------------------------------------------------------+-----------+-----------------+---------+-----------
  1 | ->  Row Adapter                                                                        |       100 |                 |      20 | 911254.47
  2 |    ->  Vector Limit                                                                    |       100 |                 |      20 | 911254.47
  3 |       ->  Vector Streaming (type: GATHER)                                              |      2400 |                 |      20 | 911325.75
  4 |          ->  Vector Limit                                                              |      2400 | 1MB             |      20 | 911247.62
  5 |             ->  Vector Sort                                                            |   3684816 | 16MB            |      20 | 911631.21
  6 |                ->  Vector Hash Join (7,29)                                             |   3684817 | 41MB(12374MB)   |      20 | 905379.41
  7 |                   ->  Vector Streaming(type: REDISTRIBUTE)                             |   3684817 | 384KB           |       4 | 883010.31
  8 |                      ->  Vector Hash Join (9,19)                                       |   3684817 | 16MB            |       4 | 861302.05
  9 |                         ->  Vector Hash Join (10,18)                                   |  11054450 | 16MB            |      44 | 427109.71
 10 |                            ->  Vector Hash Aggregate                                   |  50247501 | 397MB(12671MB)  |      54 | 395302.57
 11 |                               ->  Vector Streaming(type: REDISTRIBUTE)                 |  50247501 | 384KB           |      22 | 358663.76
 12 |                                  ->  Vector Hash Join (13,15)                          |  50247501 | 16MB            |      22 | 294300.51
 13 |                                     ->  Vector Partition Iterator                      | 287999764 | 1MB             |      26 | 227383.99
 14 |                                        ->  Partitioned CStore Scan on store_returns    | 287999764 | 1MB             |      26 | 227383.99
 15 |                                     ->  Vector Streaming(type: BROADCAST)              |      8712 | 384KB           |       4 | 975.56
 16 |                                        ->  Vector Partition Iterator                   |       363 | 1MB             |       4 | 910.65
 17 |                                           ->  Partitioned CStore Scan on date_dim      |       363 | 1MB             |       4 | 910.65
 18 |                            ->  CStore Scan on store                                    |        44 | 1MB             |       4 | 1006.39
 19 |                         ->  Vector Hash Aggregate                                      |       192 | 16MB            |      68 | 426707.38
 20 |                            ->  Vector Subquery Scan on ctr2                            |  50247501 | 1MB             |      36 | 416239.03
 21 |                               ->  Vector Hash Aggregate                                |  50247501 | 397MB(12671MB)  |      54 | 395302.57
 22 |                                  ->  Vector Streaming(type: REDISTRIBUTE)              |  50247501 | 384KB           |      22 | 358663.76
 23 |                                     ->  Vector Hash Join (24,26)                       |  50247501 | 16MB            |      22 | 294300.51
 24 |                                        ->  Vector Partition Iterator                   | 287999764 | 1MB             |      26 | 227383.99
 25 |                                           ->  Partitioned CStore Scan on store_returns | 287999764 | 1MB             |      26 | 227383.99
 26 |                                        ->  Vector Streaming(type: BROADCAST)           |      8712 | 384KB           |       4 | 975.56
 27 |                                           ->  Vector Partition Iterator                |       363 | 1MB             |       4 | 910.65
 28 |                                              ->  Partitioned CStore Scan on date_dim   |       363 | 1MB             |       4 | 910.65
 29 |                   ->  CStore Scan on customer                                          |  12000000 | 1MB             |      24 | 12923.00
(29 rows)
```

Specify the hints of HashAgg in the inner **with** clause and of the outer Hash Join. The query containing hints is as follows:

```
explain
with customer_total_return as
(select /*+ skew(store_returns(sr_store_sk sr_customer_sk)) */sr_customer_sk as ctr_customer_sk
,sr_store_sk as ctr_store_sk
,sum(SR_FEE) as ctr_total_return
from store_returns
,date_dim
where sr_returned_date_sk = d_date_sk
and d_year =2000
group by sr_customer_sk
,sr_store_sk)
 select  /*+ skew(ctr1(ctr_customer_sk)(11))*/  c_customer_id
from customer_total_return ctr1
,store
,customer
where ctr1.ctr_total_return > (select avg(ctr_total_return)*1.2
from customer_total_return ctr2
where ctr1.ctr_store_sk = ctr2.ctr_store_sk)
and s_store_sk = ctr1.ctr_store_sk
and s_state = 'NM'
and ctr1.ctr_customer_sk = c_customer_sk
order by c_customer_id
limit 100;
```

The hints indicate that the **group by** in the inner **with** clause contains skew data during redistribution by HashAgg, corresponding to the original Hash Agg operators 10 and 21; and that the **ctr_customer_sk** column in the outer **ctr1** table contains skew data during redistribution by Hash Join, corresponding to operator 6 in the original plan. The optimized plan is as follows:



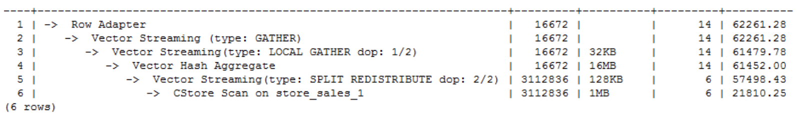
```
 id |                                   operation                                    | E-rows    | E-memory        | E-width | E-costs
----+--------------------------------------------------------------------------------+-----------+-----------------+---------+------------
  1 | ->  Row Adapter                                                                |       100 |                 |      20 | 1061778.14
  2 |   ->  Vector Limit                                                             |       100 |                 |      20 | 1061778.14
  3 |     ->  Vector Streaming (type: GATHER)                                        |      2400 |                 |      20 | 1061849.41
  4 |       ->  Vector Limit                                                         |      2400 | 1MB             |      20 | 1061771.29
  5 |         ->  Vector Sort                                                        |   3684816 | 16MB            |      20 | 1062154.87
  6 |           ->  Vector Hash Join (7,31)                                          |   3684817 | 41MB(12344MB)   |      20 | 1055903.08
  7 |             ->  Vector Streaming(type: PART REDISTRIBUTE PART ROUNDROBIN)      |   3684817 | 384KB           |       4 | 1013056.49
  8 |               ->  Vector Hash Join (9,20)                                      |   3684817 | 16MB            |       4 | 1000006.10
  9 |                 ->  Vector Hash Join (10,19)                                   |  11054450 | 16MB            |      44 | 496461.73
 10 |                   ->  Vector Hash Aggregate                                    |  50247501 | 397MB(12010MB)  |      54 | 464654.59
 11 |                     ->  Vector Streaming(type: REDISTRIBUTE)                   |  50247501 | 384KB           |      54 | 428015.79
 12 |                       ->  Vector Hash Aggregate                                |  50247501 | 397MB(12010MB)  |      54 | 330939.31
 13 |                         ->  Vector Hash Join (14,16)                           |  50247501 | 16MB            |      22 | 294300.51
 14 |                           ->  Vector Partition Iterator                        | 287999764 | 1MB             |      26 | 227383.99
 15 |                             ->  Partitioned CStore Scan on store_returns       | 287999764 | 1MB             |      26 | 227383.99
 16 |                           ->  Vector Streaming(type: BROADCAST)                |      8712 | 384KB           |       4 | 975.56
 17 |                             ->  Vector Partition Iterator                      |       363 | 1MB             |       4 | 910.65
 18 |                               ->  Partitioned CStore Scan on date_dim          |       363 | 1MB             |       4 | 910.65
 19 |                   ->  CStore Scan on store                                     |        44 | 1MB             |       4 | 1006.39
 20 |                 ->  Vector Hash Aggregate                                      |       192 | 16MB            |      68 | 496059.40
 21 |                   ->  Vector Subquery Scan on ctr2                             |  50247501 | 1MB             |      36 | 485591.05
 22 |                     ->  Vector Hash Aggregate                                  |  50247501 | 397MB(12010MB)  |      54 | 464654.59
 23 |                       ->  Vector Streaming(type: REDISTRIBUTE)                 |  50247501 | 384KB           |      54 | 428015.79
 24 |                         ->  Vector Hash Aggregate                              |  50247501 | 397MB(12010MB)  |      54 | 330939.31
 25 |                           ->  Vector Hash Join (26,28)                         |  50247501 | 16MB            |      22 | 294300.51
 26 |                             ->  Vector Partition Iterator                      | 287999764 | 1MB             |      26 | 227383.99
 27 |                               ->  Partitioned CStore Scan on store_returns     | 287999764 | 1MB             |      26 | 227383.99
 28 |                             ->  Vector Streaming(type: BROADCAST)              |      8712 | 384KB           |       4 | 975.56
 29 |                               ->  Vector Partition Iterator                    |       363 | 1MB             |       4 | 910.65
 30 |                                 ->  Partitioned CStore Scan on date_dim        |       363 | 1MB             |       4 | 910.65
 31 |             ->  Vector Streaming(type: PART LOCAL PART BROADCAST)              |  12000000 | 384KB           |      24 | 34485.50
 32 |               ->  CStore Scan on customer                                      |  12000000 | 1MB             |      24 | 12923.00
(32 rows)
```

To solve data skew in the redistribution, Hash Agg is changed to double-level Agg operators and the redistribution operators used by Hash Join are changed in the optimized plan.

- Modify the query and then specify hints.

  For example, the original query and its plan are as follows:

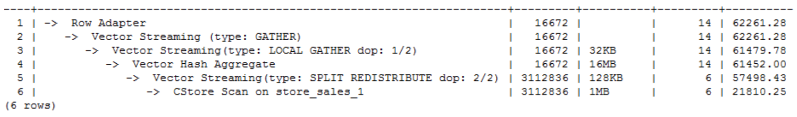
```
explain select count(*) from store_sales_1 group by round(ss_list_price);
```



```
----+------------------------------------------------------------------+---------+----------+---------+----------
  1 | ->  Row Adapter                                                  |   16672 |          |      14 | 62261.28
  2 |   ->  Vector Streaming (type: GATHER)                            |   16672 |          |      14 | 62261.28
  3 |     ->  Vector Streaming(type: LOCAL GATHER dop: 1/2)            |   16672 | 32KB     |      14 | 61479.78
  4 |       ->  Vector Hash Aggregate                                  |   16672 | 16MB     |      14 | 61452.00
  5 |         ->  Vector Streaming(type: SPLIT REDISTRIBUTE dop: 2/2)  | 3112836 | 128KB    |       6 | 57498.43
  6 |           ->  CStore Scan on store_sales_1                       | 3112836 | 1MB      |       6 | 21810.25
(6 rows)
```

Columns in hints do not support expressions. To specify hints, rewrite the query as several subqueries. The rewritten query and its plan are as follows:

```
explain
select count(*)
from (select round(ss_list_price),ss_hdemo_sk
from store_sales_1)tmp(a,ss_hdemo_sk)
group by a;
```

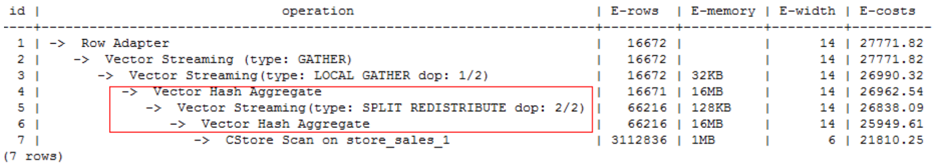


```
----+------------------------------------------------------------------+---------+----------+---------+----------
  1 | ->  Row Adapter                                                  |   16672 |          |      14 | 62261.28
  2 |   ->  Vector Streaming (type: GATHER)                            |   16672 |          |      14 | 62261.28
  3 |     ->  Vector Streaming(type: LOCAL GATHER dop: 1/2)            |   16672 | 32KB     |      14 | 61479.78
  4 |       ->  Vector Hash Aggregate                                  |   16672 | 16MB     |      14 | 61452.00
  5 |         ->  Vector Streaming(type: SPLIT REDISTRIBUTE dop: 2/2)  | 3112836 | 128KB    |       6 | 57498.43
  6 |           ->  CStore Scan on store_sales_1                       | 3112836 | 1MB      |       6 | 21810.25
(6 rows)
```

Ensure that the service logic is not changed during the rewriting.

Specify hints in the rewritten query as follows:

```
explain
select /*+ skew(tmp(a)) */ count(*)
from (select round(ss_list_price),ss_hdemo_sk
from store_sales_1)tmp(a,ss_hdemo_sk)
group by a;
```



```
 id |                          operation                                | E-rows  | E-memory | E-width | E-costs
----+-------------------------------------------------------------------+---------+----------+---------+----------
  1 | ->  Row Adapter                                                   |   16672 |          |      14 | 27771.82
  2 |   ->  Vector Streaming (type: GATHER)                             |   16672 |          |      14 | 27771.82
  3 |     ->  Vector Streaming(type: LOCAL GATHER dop: 1/2)             |   16672 | 32KB     |      14 | 26990.32
  4 |       ->  Vector Hash Aggregate                                   |   16671 | 16MB     |      14 | 26962.54
  5 |         ->  Vector Streaming(type: SPLIT REDISTRIBUTE dop: 2/2)   |   66216 | 128KB    |      14 | 26838.09
  6 |           ->  Vector Hash Aggregate                               |   66216 | 16MB     |      14 | 25949.61
  7 |             ->  CStore Scan on store_sales_1                      | 3112836 | 1MB      |       6 | 21810.25
(7 rows)
```

The plan shows that after Hash Agg is changed to double-layer Agg operators, redistributed data is greatly reduced and redistribution time shortened.

You can specify hints in columns in a subquery, for example:

```
explain
select /*+ skew(tmp(b)) */ count(*)
from (select round(ss_list_price) b,ss_hdemo_sk
from store_sales_1)tmp(a,ss_hdemo_sk)
group by a;
```

13.4.8.9 Configuration Parameter Hints

Function

A hint, or a GUC hint, specifies a configuration parameter value when a plan is generated.

Syntax

```
set [global](guc_name guc_value)
```

Parameters

- **global** indicates that the parameter set by hint takes effect at the statement level. If **global** is not specified, the parameter takes effect only in the subquery where the hint is located.
- **guc_name** indicates the name of the configuration parameter specified by hint.
- **guc_value** indicates the value of a configuration parameter specified by hint.

NOTE

- If a parameter set by hint takes effect at the statement level, the hint must be written to the top-level query instead of the subquery. For **UNION**, **INTERSECT**, **EXCEPT**, and **MINUS** statements, you can write the GUC hint at the statement level to any **SELECT** clause that participates in the set operation. The configuration parameters set by the GUC hint take effect on each **SELECT** clause that participates in the set operation.
- When a subquery is pulled up, all GUC hints on the subquery are discarded.
- If a parameter is set by both the statement-level GUC hint and the subquery-level GUC hint, the subquery-level GUC hint takes effect in the corresponding subquery, and the statement-level GUC hint takes effect in other subqueries of the statement.

Currently, GUC hints support only some configuration parameters. Some parameters cannot be configured at the subquery level and can only be configured at the statement level. The following table lists the supported parameters.

Table 13-14 Configuration parameters supported by GUC hints

Parameter	Configured at the Subquery Level (Yes/No)
agg_redistribute_enhancement	Yes
best_agg_plan	Yes
cost_model_version	No
cost_param	No
enable_bitmapscan	Yes

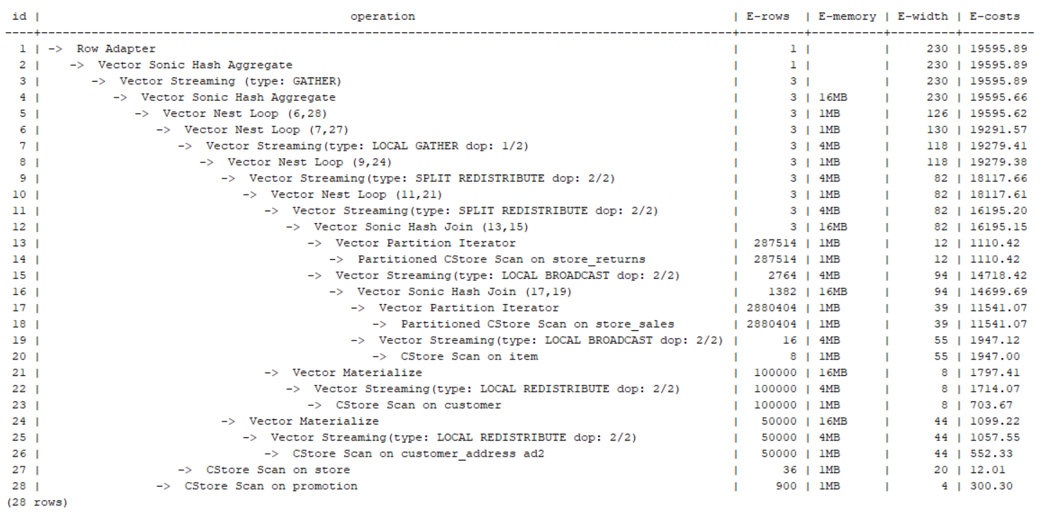
Parameter	Configured at the Subquery Level (Yes/No)
enable_broadcast	Yes
enable_extrapolation_stats	Yes
enable_fast_query_shipping	No
enable_force_vector_engine	No
enable_hashagg	Yes
enable_hashjoin	Yes
enable_index_nestloop	Yes
enable_indexscan	Yes
enable_join_pseudoconst	Yes
enable_nestloop	Yes
enable_nodegroup_debug	No
enable_partition_dynamic_pruning	Yes
enable_sort	Yes
enable_vector_engine	No
expected_computing_nodegroup	No
force_bitmapand	Yes
from_collapse_limit	Yes
join_collapse_limit	Yes
join_num_distinct	Yes
qrw_inlist2join_optmode	Yes
qual_num_distinct	Yes
query_dop	No
rewrite_rule	No
skew_option	Yes

Examples

Hint the query plan in **Examples** as follows:

```
explain
select /*+ set global(query_dop 0) */ i_product_name product_name
...
```

This hint indicates that the **query_dop** parameter is set to **0** when the plan for a statement is generated, which means the SMP adaptation function is enabled. The generated plan is as follows:



```
 id |                                 operation                                  | E-rows  | E-memory | E-width | E-costs
----+----------------------------------------------------------------------------+---------+----------+---------+---------
  1 | ->  Row Adapter                                                            |       1 |          |     230 | 19595.89
  2 |   ->  Vector Sonic Hash Aggregate                                          |       1 |          |     230 | 19595.89
  3 |     ->  Vector Streaming (type: GATHER)                                    |       3 |          |     230 | 19595.89
  4 |       ->  Vector Sonic Hash Aggregate                                      |       3 | 16MB     |     230 | 19595.66
  5 |         ->  Vector Nest Loop (6,28)                                        |       3 | 1MB      |     126 | 19595.62
  6 |           ->  Vector Nest Loop (7,27)                                      |       3 | 1MB      |     130 | 19291.57
  7 |             ->  Vector Streaming(type: LOCAL GATHER dop: 1/2)              |       3 | 4MB      |     118 | 19279.41
  8 |               ->  Vector Nest Loop (9,24)                                  |       3 | 1MB      |     118 | 19279.38
  9 |                 ->  Vector Streaming(type: SPLIT REDISTRIBUTE dop: 2/2)    |       3 | 4MB      |      82 | 18117.66
 10 |                   ->  Vector Nest Loop (11,21)                             |       3 | 1MB      |      82 | 18117.61
 11 |                     ->  Vector Streaming(type: SPLIT REDISTRIBUTE dop: 2/2)|       3 | 4MB      |      82 | 16195.20
 12 |                       ->  Vector Sonic Hash Join (13,15)                   |       3 | 16MB     |      82 | 16195.15
 13 |                         ->  Vector Partition Iterator                      |  287514 | 1MB      |      12 | 1110.42
 14 |                           ->  Partitioned CStore Scan on store_returns     |  287514 | 1MB      |      12 | 1110.42
 15 |                         ->  Vector Streaming(type: LOCAL BROADCAST dop: 2/2) |  2764 | 4MB      |      94 | 14718.42
 16 |                           ->  Vector Sonic Hash Join (17,19)               |    1382 | 16MB     |      94 | 14699.69
 17 |                             ->  Vector Partition Iterator                  | 2880404 | 1MB      |      39 | 11541.07
 18 |                               ->  Partitioned CStore Scan on store_sales   | 2880404 | 1MB      |      39 | 11541.07
 19 |                             ->  Vector Streaming(type: LOCAL BROADCAST dop: 2/2) |  16 | 4MB   |      55 | 1947.12
 20 |                               ->  CStore Scan on item                      |       8 | 1MB      |      55 | 1947.00
 21 |                     ->  Vector Materialize                                 |  100000 | 16MB     |       8 | 1797.41
 22 |                       ->  Vector Streaming(type: LOCAL REDISTRIBUTE dop: 2/2) | 100000 | 4MB   |       8 | 1714.07
 23 |                         ->  CStore Scan on customer                        |  100000 | 1MB      |       8 | 703.67
 24 |                 ->  Vector Materialize                                     |   50000 | 16MB     |      44 | 1099.22
 25 |                   ->  Vector Streaming(type: LOCAL REDISTRIBUTE dop: 2/2)  |   50000 | 4MB      |      44 | 1057.55
 26 |                     ->  CStore Scan on customer_address ad2                |   50000 | 1MB      |      44 | 552.33
 27 |             ->  CStore Scan on store                                       |      36 | 1MB      |      20 | 12.01
 28 |           ->  CStore Scan on promotion                                     |     900 | 1MB      |       4 | 300.30
(28 rows)
```

13.4.8.10 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

- 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 if a sublink contains multiple tables to be joined, because the table list in the sublink becomes invalid after the sublink is pulled up.
- Unsupported column types.
  - Skew hints are specified to optimize redistribution. They will be invalid if their corresponding columns do not support redistribution.
- Specified hints are not used.
  - If **hashjoin** or **mergejoin** is specified for non-equivalent joins, it will not be used.
  - If **indexscan** or **indexonlyscan** is specified for a table that does not have an index, it will not be used.
  - If **indexscan hint** or **indexonlyscan** is specified for a full-table scan or for a scan whose filtering conditions are not set on index columns, it will not be used.
  - The specified **indexonlyscan** hint is used only when the output column contains only indexes.
  - 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.
  - To generate a streaming plan, if the distribution key of a table is the same as its join key, **redistribute** specified for this table will not be used.

If the distribution key and join key are different for this table but the same for the other table in the join, **redistribute** specified for this table will be used but **broadcast** will not.

- If a hint for an **Agg** distribution column is not used, the possible causes are as follows:
  - The specified distribution key contains data types that do not support redistribution.
  - Redistribution is not required in the execution plan.
  - Wrong distribution key sequence numbers are executed.
  - For AP functions that use the GROUPING SETS and CUBE clauses, hints are not supported for distribution keys in window aggregate functions .

    NOTE

    Specifies the hint for the distribution column druing the Agg process.. This parameter is supported only by clusters of version 8.1.3.100 or later.
- If no sublink is pulled up, the specified **blockname** hint will not be used.
- For unused skew hints, the possible causes are:
  - The plan does not require redistribution.
  - The columns specified by hints contain distribution keys.
  - 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.
- For unused guc hints, the possible causes are:
  - The configuration parameter does not exist.
  - The configuration parameter is not supported by GUC hints.
  - The configuration parameter value is invalid.
  - The statement-level GUC hint is not written in the top-level query.
  - The configuration parameter set by the GUC hint at the subquery level cannot be set at the subquery level.
  - The subquery where the GUC hint is located is pulled up.

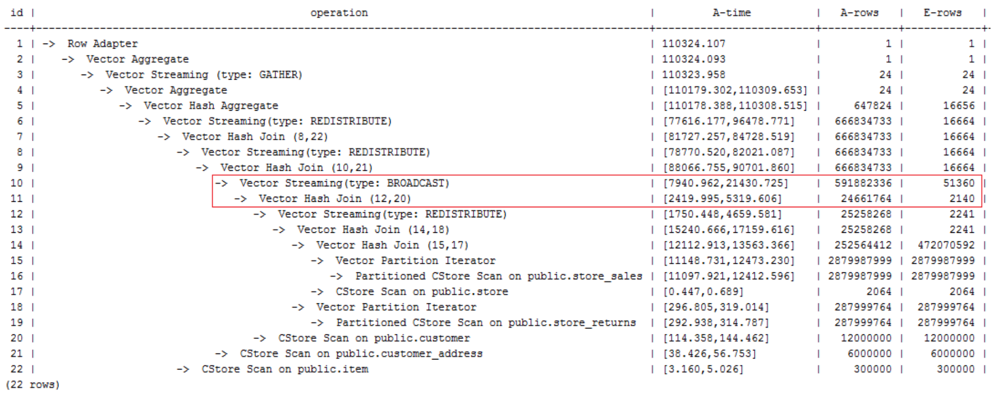
13.4.8.11 Plan Hint Cases

This section takes the statements in TPC-DS (Q24) as an example to describe how to optimize an execution plan by using hints in 1000X+24DN environments. For example:

```
select avg(netpaid) from
(select c_last_name
,c_first_name
,s_store_name
,ca_state
,s_state
,i_color
,i_current_price
,i_manager_id
,i_units
,i_size
,sum(ss_sales_price) netpaid
from store_sales
,store_returns
,store
,item
,customer
,customer_address
where ss_ticket_number = sr_ticket_number
and ss_item_sk = sr_item_sk
and ss_customer_sk = c_customer_sk
and ss_item_sk = i_item_sk
and ss_store_sk = s_store_sk
and c_birth_country = upper(ca_country)
and s_zip = ca_zip
and s_market_id=7
group by c_last_name
,c_first_name
,s_store_name
,ca_state
,s_state
,i_color
,i_current_price
,i_manager_id
,i_units
,i_size);
```

1. The original plan of this statement is as follows and the statement execution takes 110s:

Figure 13-10 Statement initial plan



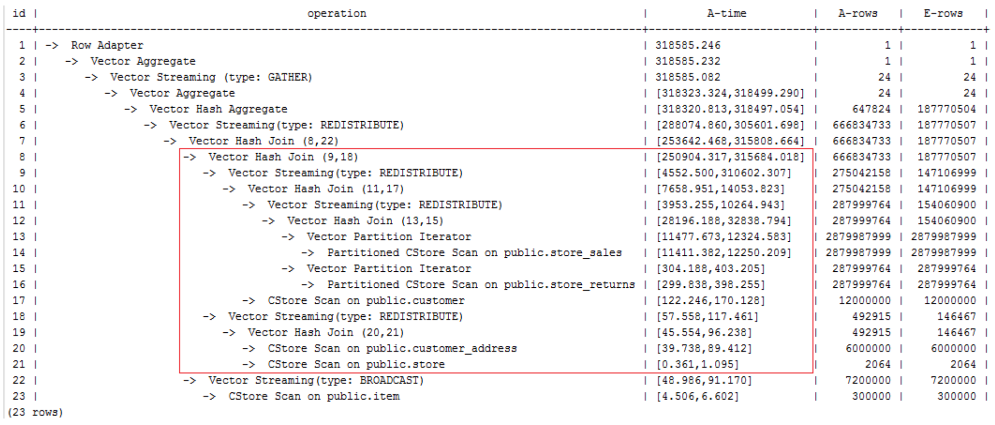
```
 id |                                   operation                                    |         A-time          |   A-rows   |   E-rows   |
----+--------------------------------------------------------------------------------+-------------------------+------------+------------+
  1 | ->  Row Adapter                                                                | 110324.107              |          1 |          1 |
  2 |   ->  Vector Aggregate                                                         | 110324.093              |          1 |          1 |
  3 |     ->  Vector Streaming (type: GATHER)                                        | 110323.958              |         24 |         24 |
  4 |       ->  Vector Aggregate                                                     | [110179.302,110309.653] |         24 |         24 |
  5 |         ->  Vector Hash Aggregate                                              | [110178.388,110308.515] |     647824 |      16656 |
  6 |           ->  Vector Streaming(type: REDISTRIBUTE)                             | [77616.177,96478.771]   |  666834733 |      16664 |
  7 |             ->  Vector Hash Join (8,22)                                        | [81727.257,84728.519]   |  666834733 |      16664 |
  8 |               ->  Vector Streaming(type: REDISTRIBUTE)                         | [78770.520,82021.087]   |  666834733 |      16664 |
  9 |                 ->  Vector Hash Join (10,21)                                   | [88066.755,90701.860]   |  666834733 |      16664 |
 10 |                   ->  Vector Streaming(type: BROADCAST)                        | [7940.962,21430.725]    |  591882336 |      51360 |
 11 |                     ->  Vector Hash Join (12,20)                               | [2419.995,5319.606]     |   24661764 |       2140 |
 12 |                       ->  Vector Streaming(type: REDISTRIBUTE)                 | [1750.448,4659.581]     |   25258268 |       2241 |
 13 |                         ->  Vector Hash Join (14,18)                           | [15240.666,17159.616]   |   25258268 |       2241 |
 14 |                           ->  Vector Hash Join (15,17)                         | [12112.913,13563.366]   |  252564412 |  472070592 |
 15 |                             ->  Vector Partition Iterator                      | [11148.731,12473.230]   | 2879987999 | 2879987999 |
 16 |                               ->  Partitioned CStore Scan on public.store_sales | [11097.921,12412.596] | 2879987999 | 2879987999 |
 17 |                             ->  CStore Scan on public.store                    | [0.447,0.689]           |       2064 |       2064 |
 18 |                           ->  Vector Partition Iterator                        | [296.805,319.014]       |  287999764 |  287999764 |
 19 |                             ->  Partitioned CStore Scan on public.store_returns | [292.938,314.787]     |  287999764 |  287999764 |
 20 |                       ->  CStore Scan on public.customer                       | [114.358,144.462]       |   12000000 |   12000000 |
 21 |                   ->  CStore Scan on public.customer_address                   | [38.426,56.753]         |    6000000 |    6000000 |
 22 |               ->  CStore Scan on public.item                                   | [3.160,5.026]           |     300000 |     300000 |
(22 rows)
```

In this plan, the performance of the layer-10 **broadcast** is poor because the estimation result generated at layer 11 is 2140 rows, which is much less than the actual number of rows. The inaccurate estimation is mainly caused by the underestimated number of rows in layer-13 hash join. In this layer, **store_sales** and **store_returns** are joined (based on the **ss_ticket_number** and **ss_item_sk** columns in **store_sales** and the **sr_ticket_number** and **sr_item_sk** columns in **store_returns**) but the multi-column correlation is not considered.

2. After the **rows** hint is used for optimization, the plan is as follows and the statement execution takes 318s:

```
select avg(netpaid) from
(select /*+rows(store_sales store_returns * 11270)*/ c_last_name ...
```

Figure 13-11 Using rows hints for optimization



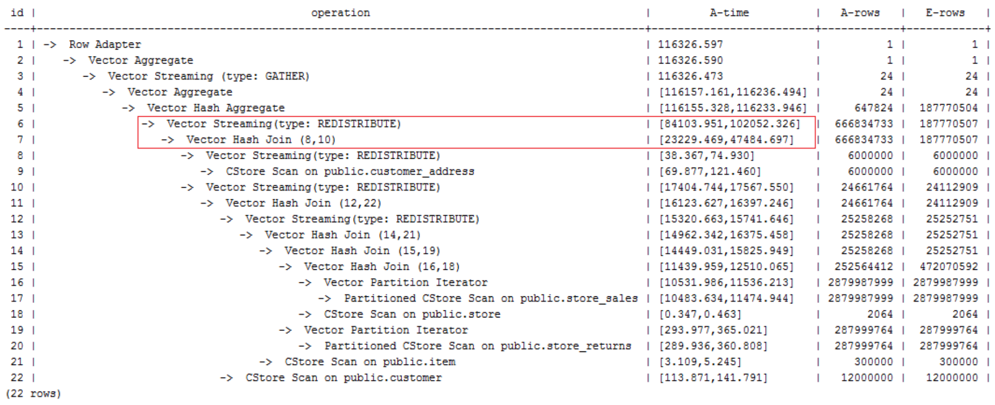
```
 id |                                    operation                                     |           A-time            |   A-rows   |   E-rows   |
----+----------------------------------------------------------------------------------+-----------------------------+------------+------------+
  1 | ->  Row Adapter                                                                  | 318585.246                  |          1 |          1 |
  2 |    ->  Vector Aggregate                                                          | 318585.232                  |          1 |          1 |
  3 |       ->  Vector Streaming (type: GATHER)                                        | 318585.082                  |         24 |         24 |
  4 |          ->  Vector Aggregate                                                    | [318323.324,318499.290]     |         24 |         24 |
  5 |             ->  Vector Hash Aggregate                                            | [318320.813,318497.054]     |     647824 |  187770504 |
  6 |                ->  Vector Streaming(type: REDISTRIBUTE)                          | [288074.860,305601.698]     |  666834733 |  187770507 |
  7 |                   ->  Vector Hash Join (8,22)                                    | [253642.468,315808.664]     |  666834733 |  187770507 |
  8 |                      ->  Vector Hash Join (9,18)                                 | [250904.317,315684.018]     |  666834733 |  187770507 |
  9 |                         ->  Vector Streaming(type: REDISTRIBUTE)                 | [4552.500,310602.307]       |  275042158 |  147106999 |
 10 |                            ->  Vector Hash Join (11,17)                          | [7658.951,14053.823]        |  275042158 |  147106999 |
 11 |                               ->  Vector Streaming(type: REDISTRIBUTE)           | [3953.255,10264.943]        |  287999764 |  154060900 |
 12 |                                  ->  Vector Hash Join (13,15)                    | [28196.188,32838.794]       |  287999764 |  154060900 |
 13 |                                     ->  Vector Partition Iterator                | [11477.673,12324.583]       | 2879987999 | 2879987999 |
 14 |                                        ->  Partitioned CStore Scan on public.store_sales | [11411.382,12250.209] | 2879987999 | 2879987999 |
 15 |                                     ->  Vector Partition Iterator                | [304.188,403.205]           |  287999764 |  287999764 |
 16 |                                        ->  Partitioned CStore Scan on public.store_returns | [299.838,398.255]   |  287999764 |  287999764 |
 17 |                               ->  CStore Scan on public.customer                 | [122.246,170.128]           |   12000000 |   12000000 |
 18 |                         ->  Vector Streaming(type: REDISTRIBUTE)                 | [57.558,117.461]            |     492915 |     146467 |
 19 |                            ->  Vector Hash Join (20,21)                          | [45.554,96.238]             |     492915 |     146467 |
 20 |                               ->  CStore Scan on public.customer_address         | [39.738,89.412]             |    6000000 |    6000000 |
 21 |                               ->  CStore Scan on public.store                    | [0.361,1.095]               |       2064 |       2064 |
 22 |                      ->  Vector Streaming(type: BROADCAST)                       | [48.986,91.170]             |    7200000 |    7200000 |
 23 |                         ->  CStore Scan on public.item                           | [4.506,6.602]               |     300000 |     300000 |
(23 rows)
```

The execution takes a longer time because layer-9 **redistribute** is slow. Considering that data skew does not occur at layer-9 **redistribute**, the slow redistribution is caused by the slow layer-8 **hashjoin** due to data skew at layer-18 **redistribute**.

3. Data skew occurs because **customer_address** has a few different values in its two join keys. Therefore, plan **customer_address** as the last one to be joined. After the hint is used for optimization, the plan is as follows and the statement execution takes 116s:

```
select avg(netpaid) from
(select /*+rows(store_sales store_returns *11270)
leading((store_sales store_returns store item customer) customer_address)*/
c_last_name ...
```

Figure 13-12 Hint optimization



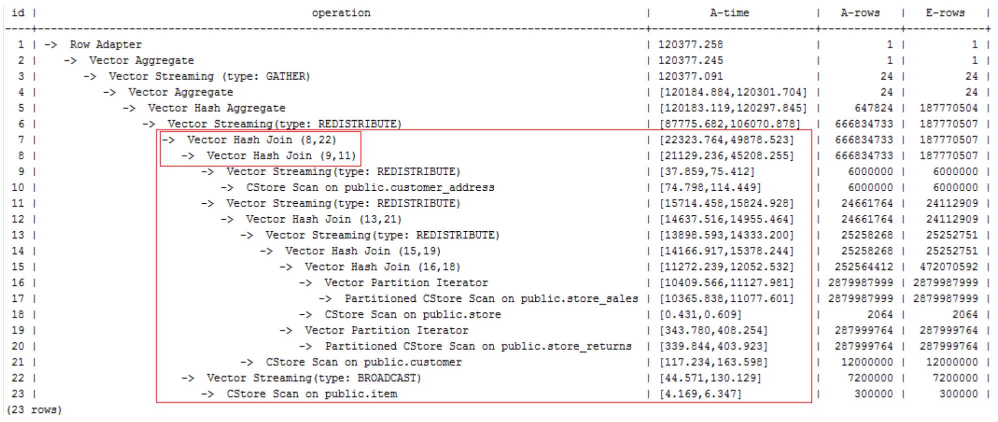
```
 id |                                    operation                                     |           A-time            |   A-rows   |   E-rows   |
----+----------------------------------------------------------------------------------+-----------------------------+------------+------------+
  1 | ->  Row Adapter                                                                  | 116326.597                  |          1 |          1 |
  2 |    ->  Vector Aggregate                                                          | 116326.590                  |          1 |          1 |
  3 |       ->  Vector Streaming (type: GATHER)                                        | 116326.473                  |         24 |         24 |
  4 |          ->  Vector Aggregate                                                    | [116157.161,116236.494]     |         24 |         24 |
  5 |             ->  Vector Hash Aggregate                                            | [116155.328,116233.946]     |     647824 |  187770504 |
  6 |                ->  Vector Streaming(type: REDISTRIBUTE)                          | [84103.951,102052.326]      |  666834733 |  187770507 |
  7 |                   ->  Vector Hash Join (8,10)                                    | [23229.469,47484.697]       |  666834733 |  187770507 |
  8 |                      ->  Vector Streaming(type: REDISTRIBUTE)                    | [38.367,74.930]             |    6000000 |    6000000 |
  9 |                         ->  CStore Scan on public.customer_address               | [69.877,121.460]            |    6000000 |    6000000 |
 10 |                      ->  Vector Streaming(type: REDISTRIBUTE)                    | [17404.744,17567.550]       |   24661764 |   24112909 |
 11 |                         ->  Vector Hash Join (12,22)                             | [16123.627,16397.246]       |   24661764 |   24112909 |
 12 |                            ->  Vector Streaming(type: REDISTRIBUTE)              | [15320.663,15741.646]       |   25258268 |   25252751 |
 13 |                               ->  Vector Hash Join (14,21)                       | [14962.342,16375.458]       |   25258268 |   25252751 |
 14 |                                  ->  Vector Hash Join (15,19)                    | [14449.031,15825.949]       |   25258268 |   25252751 |
 15 |                                     ->  Vector Hash Join (16,18)                 | [11439.959,12510.065]       |  252564412 |  472070592 |
 16 |                                        ->  Vector Partition Iterator             | [10531.986,11536.213]       | 2879987999 | 2879987999 |
 17 |                                           ->  Partitioned CStore Scan on public.store_sales | [10483.634,11474.944] | 2879987999 | 2879987999 |
 18 |                                        ->  CStore Scan on public.store           | [0.347,0.463]               |       2064 |       2064 |
 19 |                                     ->  Vector Partition Iterator                | [293.977,365.021]           |  287999764 |  287999764 |
 20 |                                        ->  Partitioned CStore Scan on public.store_returns | [289.936,360.808]   |  287999764 |  287999764 |
 21 |                                  ->  CStore Scan on public.item                  | [3.109,5.245]               |     300000 |     300000 |
 22 |                            ->  CStore Scan on public.customer                    | [113.871,141.791]           |   12000000 |   12000000 |
(22 rows)
```

Most of the time is spent on layer-6 **redistribute**. The plan needs to be further optimized.

4. The last layer redistribute contains skew. Therefore, it takes a long time. To avoid the data skew, plan the **item** table as the last one to be joined because the number of rows is not reduced after **item** is joined. After the hint is used for optimization, the plan is as follows and the statement execution takes 120s:

```
select avg(netpaid) from
(select /*+rows(store_sales store_returns *11270)
leading((customer_address (store_sales store_returns store customer) item))
c_last_name ...
```

Figure 13-13 Modifying hints and executing statements



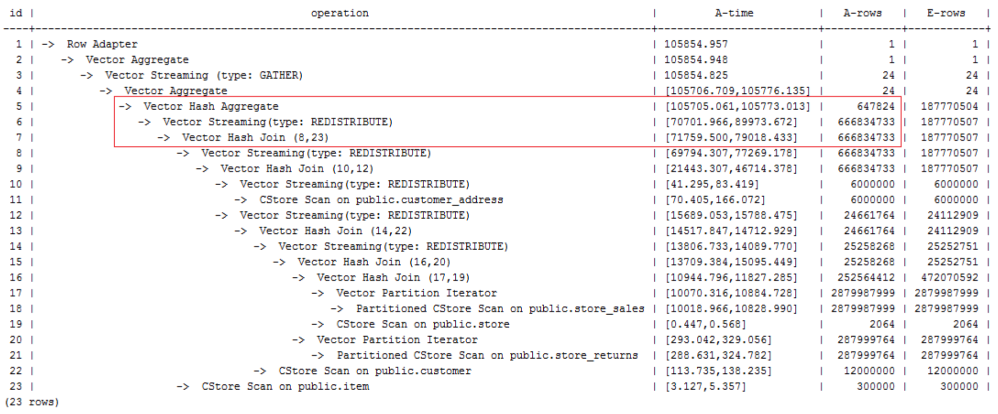
```
 id |                                 operation                                 |         A-time          |  A-rows    |  E-rows    |
----+---------------------------------------------------------------------------+-------------------------+------------+------------+
  1 | ->  Row Adapter                                                           | 120377.258              |          1 |          1 |
  2 |   ->  Vector Aggregate                                                    | 120377.245              |          1 |          1 |
  3 |     ->  Vector Streaming (type: GATHER)                                   | 120377.091              |         24 |         24 |
  4 |       ->  Vector Aggregate                                                | [120184.884,120301.704] |         24 |         24 |
  5 |         ->  Vector Hash Aggregate                                         | [120183.119,120297.845] |     647824 |  187770504 |
  6 |           ->  Vector Streaming(type: REDISTRIBUTE)                        | [87775.682,106070.878]  |  666834733 |  187770507 |
  7 |             ->  Vector Hash Join (8,22)                                   | [22323.764,49878.523]   |  666834733 |  187770507 |
  8 |               ->  Vector Hash Join (9,11)                                 | [21129.236,45208.255]   |  666834733 |  187770507 |
  9 |                 ->  Vector Streaming(type: REDISTRIBUTE)                  | [37.859,75.412]         |    6000000 |    6000000 |
 10 |                   ->  CStore Scan on public.customer_address              | [74.798,114.449]        |    6000000 |    6000000 |
 11 |                 ->  Vector Streaming(type: REDISTRIBUTE)                  | [15714.458,15824.928]   |   24661764 |   24112909 |
 12 |                   ->  Vector Hash Join (13,21)                            | [14637.516,14955.464]   |   24661764 |   24112909 |
 13 |                     ->  Vector Streaming(type: REDISTRIBUTE)              | [13898.593,14333.200]   |   25258268 |   25252751 |
 14 |                       ->  Vector Hash Join (15,19)                        | [14166.917,15378.244]   |   25258268 |   25252751 |
 15 |                         ->  Vector Hash Join (16,18)                      | [11272.239,12052.532]   |  252564412 |  472070592 |
 16 |                           ->  Vector Partition Iterator                   | [10409.566,11127.981]   | 2879987999 | 2879987999 |
 17 |                             ->  Partitioned CStore Scan on public.store_sales | [10365.838,11077.601] | 2879987999 | 2879987999 |
 18 |                           ->  CStore Scan on public.store                 | [0.431,0.609]           |       2064 |       2064 |
 19 |                         ->  Vector Partition Iterator                     | [343.780,408.254]       |  287999764 |  287999764 |
 20 |                           ->  Partitioned CStore Scan on public.store_returns | [339.844,403.923]   |  287999764 |  287999764 |
 21 |                     ->  CStore Scan on public.customer                    | [117.234,163.598]       |   12000000 |   12000000 |
 22 |               ->  Vector Streaming(type: BROADCAST)                       | [44.571,130.129]        |    7200000 |    7200000 |
 23 |                 ->  CStore Scan on public.item                            | [4.169,6.347]           |     300000 |     300000 |
(23 rows)
```

Data skew occurs after the join of **item** and **customer_address** because **item** is broadcasted at layer-22. As a result, layer-6 **redistribute** is still slow.

5. Add a hint to disable **broadcast** for **item** or add a **redistribute** hint for the join result of **item** and **customer_address**. After the hint is used for optimization, the plan is as follows and the statement execution takes 105s:

```
select avg(netpaid) from
(select /*+rows(store_sales store_returns *11270)
leading((customer_address (store_sales store_returns store customer) item))
no broadcast(item)*/
c_last_name ...
```

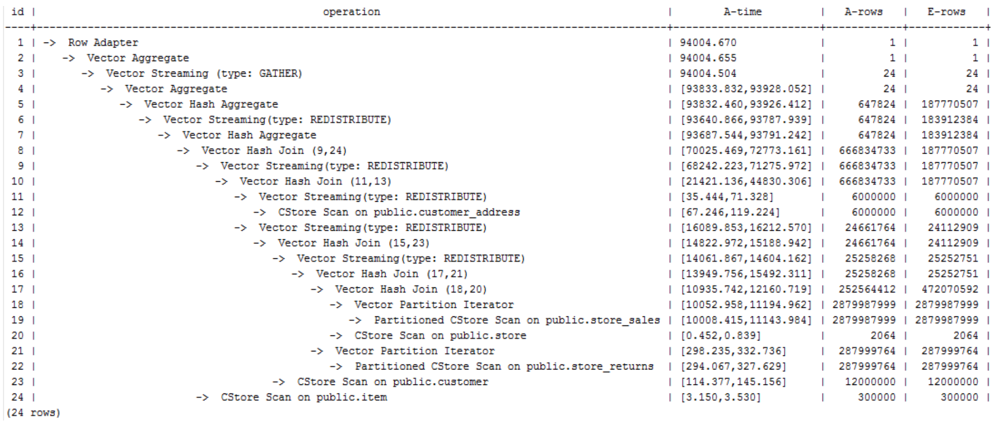
Figure 13-14 Execution plan



```
 id |                                 operation                                 |         A-time          |  A-rows    |  E-rows    |
----+---------------------------------------------------------------------------+-------------------------+------------+------------+
  1 | ->  Row Adapter                                                           | 105854.957              |          1 |          1 |
  2 |   ->  Vector Aggregate                                                    | 105854.948              |          1 |          1 |
  3 |     ->  Vector Streaming (type: GATHER)                                   | 105854.825              |         24 |         24 |
  4 |       ->  Vector Aggregate                                                | [105706.709,105776.135] |         24 |         24 |
  5 |         ->  Vector Hash Aggregate                                         | [105705.061,105773.013] |     647824 |  187770504 |
  6 |           ->  Vector Streaming(type: REDISTRIBUTE)                        | [70701.966,89973.672]   |  666834733 |  187770507 |
  7 |             ->  Vector Hash Join (8,23)                                   | [71759.500,79018.433]   |  666834733 |  187770507 |
  8 |               ->  Vector Streaming(type: REDISTRIBUTE)                    | [69794.307,77269.178]   |  666834733 |  187770507 |
  9 |                 ->  Vector Hash Join (10,12)                              | [21443.307,46714.378]   |  666834733 |  187770507 |
 10 |                   ->  Vector Streaming(type: REDISTRIBUTE)                | [41.295,83.419]         |    6000000 |    6000000 |
 11 |                     ->  CStore Scan on public.customer_address            | [70.405,166.072]        |    6000000 |    6000000 |
 12 |                   ->  Vector Streaming(type: REDISTRIBUTE)                | [15689.053,15788.475]   |   24661764 |   24112909 |
 13 |                     ->  Vector Hash Join (14,22)                          | [14517.847,14712.929]   |   24661764 |   24112909 |
 14 |                       ->  Vector Streaming(type: REDISTRIBUTE)            | [13806.733,14089.770]   |   25258268 |   25252751 |
 15 |                         ->  Vector Hash Join (16,20)                      | [13709.384,15095.449]   |   25258268 |   25252751 |
 16 |                           ->  Vector Hash Join (17,19)                    | [10944.796,11827.285]   |  252564412 |  472070592 |
 17 |                             ->  Vector Partition Iterator                 | [10070.316,10884.728]   | 2879987999 | 2879987999 |
 18 |                               ->  Partitioned CStore Scan on public.store_sales | [10018.966,10828.990] | 2879987999 | 2879987999 |
 19 |                             ->  CStore Scan on public.store               | [0.447,0.568]           |       2064 |       2064 |
 20 |                           ->  Vector Partition Iterator                   | [293.042,329.056]       |  287999764 |  287999764 |
 21 |                             ->  Partitioned CStore Scan on public.store_returns | [288.631,324.782] |  287999764 |  287999764 |
 22 |                       ->  CStore Scan on public.customer                  | [113.735,138.235]       |   12000000 |   12000000 |
 23 |               ->  CStore Scan on public.item                              | [3.127,5.357]           |     300000 |     300000 |
(23 rows)
```

6. The last layer uses single-layer **Agg** and the number of rows is greatly reduced. Set **best_agg_plan** to **3** and change the single-layer **Agg** to a double-layer **Agg**. The plan is as follows and the statement execution takes 94s. The optimization ends.

Figure 13-15 Final optimization plan

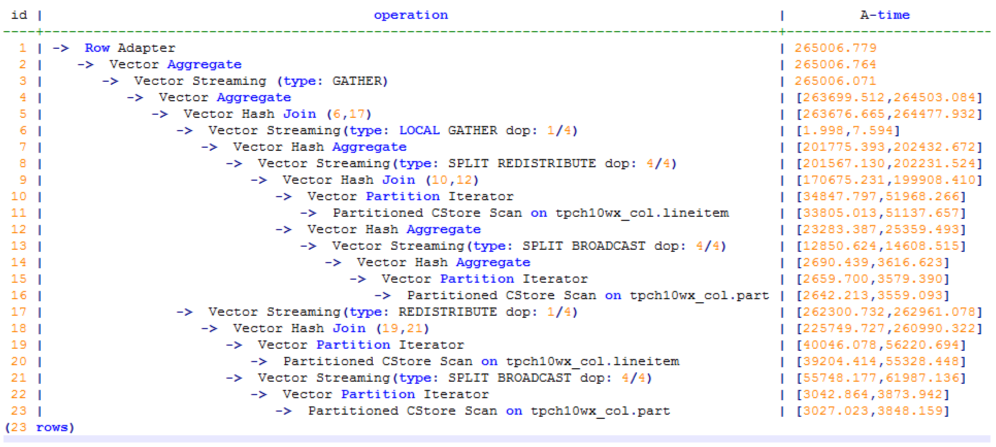


```
 id |                                   operation                                   |        A-time         |   A-rows   |   E-rows   |
----+-------------------------------------------------------------------------------+-----------------------+------------+------------+
  1 | ->  Row Adapter                                                               | 94004.670             |          1 |          1 |
  2 |   ->  Vector Aggregate                                                        | 94004.655             |          1 |          1 |
  3 |     ->  Vector Streaming (type: GATHER)                                       | 94004.504             |         24 |         24 |
  4 |       ->  Vector Aggregate                                                    | [93833.832,93928.052] |         24 |         24 |
  5 |         ->  Vector Hash Aggregate                                             | [93832.460,93926.412] |     647824 |  187770507 |
  6 |           ->  Vector Streaming(type: REDISTRIBUTE)                            | [93640.866,93787.939] |     647824 |  183912384 |
  7 |             ->  Vector Hash Aggregate                                         | [93687.544,93791.242] |     647824 |  183912384 |
  8 |               ->  Vector Hash Join (9,24)                                     | [70025.469,72773.161] |  666834733 |  187770507 |
  9 |                 ->  Vector Streaming(type: REDISTRIBUTE)                      | [68242.223,71275.972] |  666834733 |  187770507 |
 10 |                   ->  Vector Hash Join (11,13)                                | [21421.136,44830.306] |  666834733 |  187770507 |
 11 |                     ->  Vector Streaming(type: REDISTRIBUTE)                  | [35.444,71.328]       |    6000000 |    6000000 |
 12 |                       ->  CStore Scan on public.customer_address              | [67.246,119.224]      |    6000000 |    6000000 |
 13 |                     ->  Vector Streaming(type: REDISTRIBUTE)                  | [16089.853,16212.570] |   24661764 |   24112909 |
 14 |                       ->  Vector Hash Join (15,23)                            | [14822.972,15188.942] |   24661764 |   24112909 |
 15 |                         ->  Vector Streaming(type: REDISTRIBUTE)              | [14061.867,14604.162] |   25258268 |   25252751 |
 16 |                           ->  Vector Hash Join (17,21)                        | [13949.756,15492.311] |   25258268 |   25252751 |
 17 |                             ->  Vector Hash Join (18,20)                      | [10935.742,12160.719] |  252564412 |  472070592 |
 18 |                               ->  Vector Partition Iterator                   | [10052.958,11194.962] | 2879987999 | 2879987999 |
 19 |                                 ->  Partitioned CStore Scan on public.store_sales | [10008.415,11143.984] | 2879987999 | 2879987999 |
 20 |                               ->  CStore Scan on public.store                 | [0.452,0.839]         |       2064 |       2064 |
 21 |                             ->  Vector Partition Iterator                     | [298.235,332.736]     |  287999764 |  287999764 |
 22 |                               ->  Partitioned CStore Scan on public.store_returns | [294.067,327.629]  |  287999764 |  287999764 |
 23 |                         ->  CStore Scan on public.customer                    | [114.377,145.156]     |   12000000 |   12000000 |
 24 |                 ->  CStore Scan on public.item                                | [3.150,3.530]         |     300000 |     300000 |
(24 rows)
```

If the query performance deteriorates due to statistics changes, you can use hints to optimize the query plan. Take TPCH-Q17 as an example. The query performance deteriorates after the value of **default_statistics_target** is changed from the default one to **–2** for statistics collection.

1. If **default_statistics_target** is set to the default value **100**, the plan is as follows.

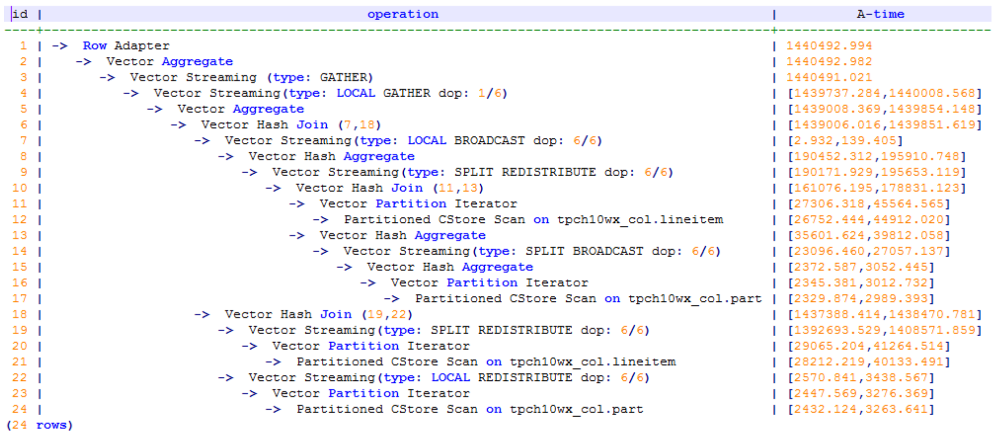
Figure 13-16 Default statistics



```
 id |                              operation                                   |         A-time
----+--------------------------------------------------------------------------+-------------------------
  1 | ->  Row Adapter                                                          | 265006.779
  2 |   ->  Vector Aggregate                                                   | 265006.764
  3 |     ->  Vector Streaming (type: GATHER)                                  | 265006.071
  4 |       ->  Vector Aggregate                                               | [263699.512,264503.084]
  5 |         ->  Vector Hash Join (6,17)                                      | [263676.665,264477.932]
  6 |           ->  Vector Streaming(type: LOCAL GATHER dop: 1/4)              | [1.998,7.594]
  7 |             ->  Vector Hash Aggregate                                    | [201775.393,202432.672]
  8 |               ->  Vector Streaming(type: SPLIT REDISTRIBUTE dop: 4/4)    | [201567.130,202231.524]
  9 |                 ->  Vector Hash Join (10,12)                             | [170675.231,199908.410]
 10 |                   ->  Vector Partition Iterator                          | [34847.797,51968.266]
 11 |                     ->  Partitioned CStore Scan on tpch10wx_col.lineitem | [33805.013,51137.657]
 12 |                   ->  Vector Hash Aggregate                              | [23283.387,25359.493]
 13 |                     ->  Vector Streaming(type: SPLIT BROADCAST dop: 4/4) | [12850.624,14608.515]
 14 |                       ->  Vector Hash Aggregate                          | [2690.439,3616.623]
 15 |                         ->  Vector Partition Iterator                    | [2659.700,3579.390]
 16 |                           ->  Partitioned CStore Scan on tpch10wx_col.part | [2642.213,3559.093]
 17 |           ->  Vector Streaming(type: REDISTRIBUTE dop: 1/4)              | [262300.732,262961.078]
 18 |             ->  Vector Hash Join (19,21)                                 | [225749.727,260990.322]
 19 |               ->  Vector Partition Iterator                              | [40046.078,56220.694]
 20 |                 ->  Partitioned CStore Scan on tpch10wx_col.lineitem     | [39204.414,55328.448]
 21 |               ->  Vector Streaming(type: SPLIT BROADCAST dop: 4/4)       | [55748.177,61987.136]
 22 |                 ->  Vector Partition Iterator                            | [3042.864,3873.942]
 23 |                   ->  Partitioned CStore Scan on tpch10wx_col.part       | [3027.023,3848.159]
(23 rows)
```

2. If **default_statistics_target** is set to **–2**, the plan is as follows.

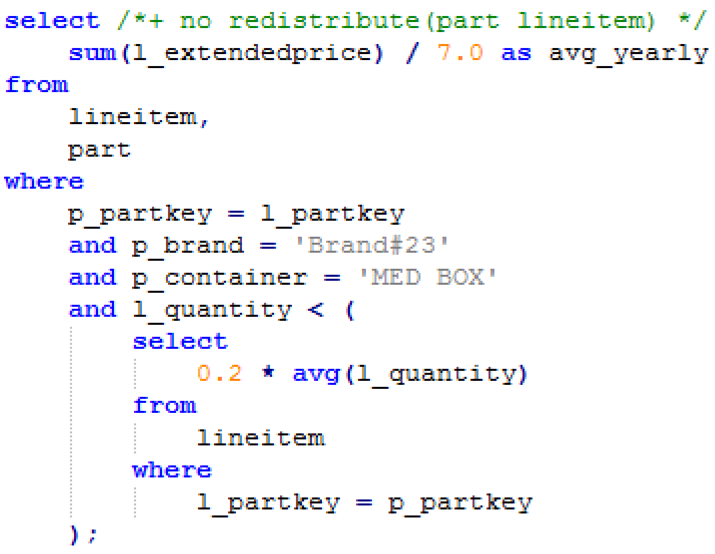
Figure 13-17 Changes in statistics



```
id |                                           operation                                        |          A-time
---+--------------------------------------------------------------------------------------------+------------------------------
 1 | ->  Row Adapter                                                                            | 1440492.994
 2 |    ->  Vector Aggregate                                                                    | 1440492.982
 3 |       ->  Vector Streaming (type: GATHER)                                                  | 1440491.021
 4 |          ->  Vector Streaming(type: LOCAL GATHER dop: 1/6)                                 | [1439737.284,1440008.568]
 5 |             ->  Vector Aggregate                                                           | [1439008.369,1439854.148]
 6 |                ->  Vector Hash Join (7,18)                                                 | [1439006.016,1439851.619]
 7 |                   ->  Vector Streaming(type: LOCAL BROADCAST dop: 6/6)                     | [2.932,139.405]
 8 |                      ->  Vector Hash Aggregate                                             | [190452.312,195910.748]
 9 |                         ->  Vector Streaming(type: SPLIT REDISTRIBUTE dop: 6/6)            | [190171.929,195653.119]
10 |                            ->  Vector Hash Join (11,13)                                    | [161076.195,178831.123]
11 |                               ->  Vector Partition Iterator                                | [27306.318,45564.565]
12 |                                  ->  Partitioned CStore Scan on tpch10wx_col.lineitem       | [26752.444,44912.020]
13 |                               ->  Vector Hash Aggregate                                    | [35601.624,39812.058]
14 |                                  ->  Vector Streaming(type: SPLIT BROADCAST dop: 6/6)      | [23096.460,27057.137]
15 |                                     ->  Vector Hash Aggregate                              | [2372.587,3052.445]
16 |                                        ->  Vector Partition Iterator                       | [2345.381,3012.732]
17 |                                           ->  Partitioned CStore Scan on tpch10wx_col.part | [2329.874,2989.393]
18 |                   ->  Vector Hash Join (19,22)                                             | [1437388.414,1438470.781]
19 |                      ->  Vector Streaming(type: SPLIT REDISTRIBUTE dop: 6/6)               | [1392693.529,1408571.859]
20 |                         ->  Vector Partition Iterator                                      | [29065.204,41264.514]
21 |                            ->  Partitioned CStore Scan on tpch10wx_col.lineitem             | [28212.219,40133.491]
22 |                      ->  Vector Streaming(type: LOCAL REDISTRIBUTE dop: 6/6)               | [2570.841,3438.567]
23 |                         ->  Vector Partition Iterator                                      | [2447.569,3276.369]
24 |                            ->  Partitioned CStore Scan on tpch10wx_col.part                | [2432.124,3263.641]
(24 rows)
```

3. After the analysis, the cause is that the stream type is changed from **BroadCast** to **Redistribute** during the join of the **lineitem** and **part** tables. You can use a hint to change the stream type back to **BroadCast**. The figure below shows an example.

Figure 13-18 Statements



```
select /*+ no redistribute(part lineitem) */
    sum(l_extendedprice) / 7.0 as avg_yearly
from
    lineitem,
    part
where
    p_partkey = l_partkey
    and p_brand = 'Brand#23'
    and p_container = 'MED BOX'
    and l_quantity < (
        select
            0.2 * avg(l_quantity)
        from
            lineitem
        where
            l_partkey = p_partkey
    );
```

13.4.9 Routinely Maintaining Tables

To ensure proper database running, after INSERT and DELETE operations, you need to routinely do **VACUUM FULL** and **ANALYZE** as appropriate for customer scenarios and update statistics to obtain better performance.

Related Concepts

You need to routinely run **VACUUM**, **VACUUM FULL**, and **ANALYZE** to maintain tables, because:

- **VACUUM FULL** reclaims disk space occupied by updated or deleted data and combines small-size data files.

- **VACUUM** maintains a visualized mapping to track pages that contain arrays visible to other active transactions. A common index scan uses the mapping to obtain the corresponding array and check whether pages are visible to the current transaction. If the array cannot be obtained, the visibility is checked by fetching stack arrays. Therefore, updating the visible mapping of a table can accelerate unique index scans.
- **VACUUM** can avoid old data loss caused by duplicate transaction IDs when the number of executed transactions exceeds the database threshold.
- **ANALYZE** collects statistics on tables in databases. The statistics are stored in the PG_STATISTIC system catalog. Then, the query optimizer uses the statistics to work out the most efficient execution plan.

Procedure

Step 1 Run the **VACUUM** or **VACUUM FULL** command to reclaim disk space.

- **VACUUM**:

  Do **VACUUM** to the table:

  ```
  VACUUM customer;
  VACUUM
  ```

  This command can be concurrently executed with database operation commands, including **SELECT**, **INSERT**, **UPDATE**, and **DELETE**; excluding **ALTER TABLE**.

  Do **VACUUM** to the partitioned table:

  ```
  VACUUM customer_par PARTITION ( P1 );
  VACUUM
  ```

- VACUUM FULL:

  ```
  VACUUM FULL customer;
  VACUUM
  ```

  VACUUM FULL needs to add exclusive locks on tables it operates on and requires that all other database operations be suspended.

  When reclaiming disk space, you can query for the session corresponding to the earliest transactions in the cluster, and then end the earliest long transactions as needed to make full use of the disk space.

  a. Run the following command to query for oldestxmin on the GTM:

  ```
  select * from pgxc_gtm_snapshot_status();
  ```

  b. Run the following command to query for the PID of the corresponding session on the CN. *xmin* is the oldestxmin obtained in the previous step.

  ```
  select * from pgxc_running_xacts() where xmin=1400202010;
  ```

Step 2 Do **ANALYZE** to update statistical information.

```
ANALYZE customer;
ANALYZE
```

Do **ANALYZE VERBOSE** to update statistics and display table information.

```
ANALYZE VERBOSE customer;
ANALYZE
```

You can use **VACUUM ANALYZE** at the same time to optimize the query.

```
VACUUM ANALYZE customer;
VACUUM
```

NOTE

VACUUM and **ANALYZE** cause a substantial increase in I/O traffic, which may cause poor performance of other active sessions. Therefore, you are advised to set by specifying the **vacuum_cost_delay** parameter.

Step 3 Delete a table

```
DROP TABLE customer;
DROP TABLE customer_par;
DROP TABLE part;
```

If the following output is displayed, the index has been deleted.

```
DROP TABLE
```

----End

Maintenance Suggestion

- Routinely do **VACUUM FULL** to large tables. If the database performance deteriorates, do **VACUUM FULL** to the entire database. If the database performance is stable, you are advised to monthly do **VACUUM FULL**.
- Routinely do **VACUUM FULL** to system catalogs, mainly **PG_ATTRIBUTE**.
- The automatic vacuum process (**AUTOVACUUM**) in the system automatically runs the **VACUUM** and **ANALYZE** statements to reclaim the record space marked as the deleted state and to update statistics related to the table.

13.4.10 Routinely Recreating an Index

Context

When data deletion is repeatedly performed in the database, index keys will be deleted from the index page, resulting in index distention. Recreating an index routinely improves query efficiency.

The database supports B-tree, GIN, and psort indexes.

- Recreating a B-tree index helps improve query efficiency.
  – If massive data is deleted, index keys on the index page will be deleted. As a result, the number of index pages reduces and index bloat occurs. Recreating an index helps reclaim wasted space.
  – In the created index, pages adjacent in its logical structure are adjacent in its physical structure. Therefore, a created index achieves higher access speed than an index that has been updated for multiple times.
- You are advised not to recreate a non-B-tree index.

Rebuilding an Index

Use either of the following two methods to recreate an index:

- Run the **DROP INDEX** statement to delete an index and run the **CREATE INDEX** statement to create an index.

  When you delete an index, a temporary exclusive lock is added in the parent table to block related read/write operations. When you create an index, the write operation is locked but the read operation is not. The data is read and scanned by order.

- Run the **REINDEX** statement to recreate an index:
  - When you run the **REINDEX TABLE** statement to recreate an index, an exclusive lock is added to block related read/write operations.
  - When you run the **REINDEX INTERNAL TABLE** statement to recreate an index for a **desc** table (), an exclusive lock is added to block read/write operations on the table.

Procedure

Assume the ordinary index areaS_idx exists in the **area_id** column of the imported table **areaS**. Use either of the following two methods to recreate an index:

- Run the **DROP INDEX** statement to delete the index and run the **CREATE INDEX** statement to create an index.
  a. Delete an index.
  ```
  DROP INDEX areaS_idx;
  DROP INDEX
  ```
  b. Create an index.
  ```
  CREATE INDEX areaS_idx ON areaS (area_id);
  CREATE INDEX
  ```
- Run the **REINDEX** statement to recreate an index.
  - Run the **REINDEX TABLE** statement to recreate an index.
  ```
  REINDEX TABLE areaS;
  REINDEX
  ```
  - Run the **REINDEX INTERNAL TABLE** statement to recreate an index for a **desc** table ().
  ```
  REINDEX INTERNAL TABLE areaS;
  REINDEX
  ```

13.4.11 Automatic Retry upon SQL Statement Execution Errors

With automatic retry (referred to as CN retry), GaussDB(DWS) retries an SQL statement when the execution of a statement fails. If an SQL statement sent from the **gsql** client, JDBC driver, or ODBC driver fails to be executed, the CN can automatically identify the error reported during execution and re-deliver the task to retry.

The restrictions of this function are as follows:

- Functionality restrictions:
  - CN retry increases execution success rate but does not guarantee success.
  - CN retry is enabled by default. In this case, the system records logs about temporary tables. If it is disabled, the system will not record the logs. Therefore, do not repeatedly enable and disable CN retry when temporary tables are used. Otherwise, data inconsistency may occur after a CN retry following a primary/standby switchover.
  - CN retry is enabled by default. In this case, the **unlogged** keyword is ignored in the statement for creating unlogged tables and thereby ordinary tables will be created by using this statement. If CN retry is disabled, the system records logs about unlogged tables. Therefore, do not repeatedly enable and disable CN retry when unlogged tables are

used. Otherwise, data inconsistency may occur after a CN retry following a primary/standby switchover.
  - When GDS is used to export data, CN retry is supported. The existing mechanism checks for duplicate files and deletes duplicate files during data export. Therefore, you are advised not to repeatedly export data for the same foreign table unless you are sure that files with the same name in the data directory need to be deleted.
- Error type restrictions:

  Only the error types in **Table 13-15** are supported.
- Statement type restrictions:

  Support single-statement CN retry, stored procedures, functions, and anonymous blocks. Statements in transaction blocks are not supported.
- Statement restrictions of a stored procedure:
  - If an error occurs during the execution of a stored procedure containing **EXCEPTION** (including statement block execution and statement execution in EXCEPTION), the stored procedure can be retried. If an internal error occurs, the stored procedure will retry first, but if the error is captured by **EXCEPTION**, the stored procedure cannot be retried.
  - Packages that use global variables are not supported.
  - **DBMS_JO** is not supported.
  - **UTL_FILE** is not supported.
  - If the stored procedure has printed information (such as **dbms_output.put_line** or **raise info**), the printed information will be output repeatedly when retry occurs, and "Notice: Retry triggered, some message may be duplicated. " will be output before the repeated information.
- Cluster status restrictions:
  - Only DNs or GTMs are faulty.
  - The cluster can be recovered before the number of CN retries reaches the allowed maximum (controlled by **max_query_retry_times**). Otherwise, CN retry may fail.
  - CN retry is not supported during scale-out.
- Data import restrictions:
  - The **COPY FROM STDIN** statement is not supported.
  - The **gsql \copy from** metacommand is not supported.
  - **JDBC CopyManager copyIn** is not supported.

Table 13-15 lists the error types supported by CN retry and the corresponding error codes. You can use the GUC parameter **retry_ecode_list** to set the list of error types supported by CN retry. You are not advised to modify this parameter. To modify it, contact the technical support.

Table 13-15 Error types supported by CN retry

Error Type	Error Code	Remarks
CONNECTION_RESET_BY_PEER	YY001	TCP communication errors: Connection reset by peer (communication between the CN and DNs)
STREAM_CONNECTION_RESET_BY_PEER	YY002	TCP communication errors: Stream connection reset by peer (communication between DNs)
LOCK_WAIT_TIMEOUT	YY003	Lock wait timeout
CONNECTION_TIMED_OUT	YY004	TCP communication errors: Connection timed out
SET_QUERY_ERROR	YY005	Failed to deliver the **SET** command: Set query
OUT_OF_LOGICAL_MEMORY	YY006	Failed to apply for memory: Out of logical memory
SCTP_MEMORY_ALLOC	YY007	SCTP communication errors: Memory allocate error
SCTP_NO_DATA_IN_BUFFER	YY008	SCTP communication errors: SCTP no data in buffer
SCTP_RELEASE_MEMORY_CLOSE	YY009	SCTP communication errors: Release memory close
SCTP_TCP_DISCONNECT	YY010	SCTP communication errors: TCP disconnect
SCTP_DISCONNECT	YY011	SCTP communication errors: SCTP disconnect
SCTP_REMOTE_CLOSE	YY012	SCTP communication errors: Stream closed by remote
SCTP_WAIT_POLL_UNKNOW	YY013	Waiting for an unknown poll: SCTP wait poll unknown
SNAPSHOT_INVALID	YY014	Snapshot invalid
ERRCODE_CONNECTION_RECEIVE_WRONG	YY015	Connection receive wrong
OUT_OF_MEMORY	53200	Out of memory
CONNECTION_FAILURE	08006	GTM errors: Connection failure

Error Type	Error Code	Remarks
CONNECTION_EXCEPTION	08000	Failed to communicate with DNs due to connection errors: Connection exception
ADMIN_SHUTDOWN	57P01	System shutdown by administrators: Admin shutdown
STREAM_REMOTE_CLOSE_SOCKET	XX003	Remote socket disabled: Stream remote close socket
ERRCODE_STREAM_DUPLICATE_QUERY_ID	XX009	Duplicate query id
ERRCODE_STREAM_CONCURRENT_UPDATE	YY016	Stream concurrent update
ERRCODE_LLVM_BAD_ALLOC_ERROR	CG003	Memory allocation error: Allocate error
ERRCODE_LLVM_FATAL_ERROR	CG004	Fatal error
HashJoin temporary file reading error (ERRCODE_HASHJOIN_TEMP_FILE_ERROR).	F0011	File error
Partition number error (ERRCODE_PARTITION_NUM_CHANGED).	45003	During scanning on a list partition table, it is found that the number of partitions is different from that in the optimization phase. This problem usually occurs when the queries and **ADD**/**DROP** partitions are concurrently executed. (This error is supported only by cluster 8.1.3 and later versions.)

To enable CN retry, set the following GUC parameters:

- Mandatory GUC parameters (required by both CNs and DNs)

  max_query_retry_times

  CAUTION

  If CN retry is enabled, temporary table data is logged. For data consistency, do not switch the enabled/disabled status for CN retry when the temporary tables are being used by sessions.

- Optional GUC parameters

  cn_send_buffer_size

max_cn_temp_file_size

13.4.12 query_band Load Identification

Overview

GaussDB(DWS) implements load identification and intra-queue priority control based on query_band. It provides more flexible load identification methods and identifies load queues based on job types, application names, and script names. Users can flexibly configure query_band identification queues based on service scenarios. In addition, priority control of job delivery in the queue is implemented. In the future, priority control of resources in the queue will be gradually implemented.

Administrators can configure the queue associated with query_band and estimate the memory limit based on service scenarios and job types to implement more flexible load control and resource management and control. If query_band is not configured for the service or the user does not associate query_band with an action, the queue associated with the user and the priority in the queue is used by default.

Load Behaviors Supported by query_band

query_band is a session-level GUC parameter. It is a job identifier of the character data type. Its value can be any string. However, for easier differentiation and configuration, query_band only identifies key-value pairs. For example:

```
SET query_band='JobName=abc;AppName=test;UserName=user';
```

JobName=abc, **AppName=test**, and **UserName=user** are independent key-value pairs. Specifications of the query_band key-value pairs:

- query_band is set in key-value pair mode, that is, 'key=value'. Multiple query_band key-value pairs can be set in a session. Multiple key-value pairs are separated by semicolons (;). The maximum length of both the **query_band** key-value pair and parameter value is 1,024 characters.
- The query_band key-value pair supports the following valid characters: digits 0 to 9, uppercase letters A to Z, lowercase letters a to z, '.', '-', '_', and '#'.

query_band is configured, and identifies load behaviors, using key-value pairs. The supported load behaviors are described in **Table 13-16**.

Table 13-16 Load behaviors supported by QUERY_BAND

Type	Behavior	Behavior Description
Workload management (workload)	Resource pool (respool)	query_band associated with a resource pool
Workload management (workload)	Priority	Priority in the queue

Type	Behavior	Behavior Description
Order	Queue (respool) Currently, this field is invalid and is used for future extension.	**query_band** query order

The "Type" is used to classify load behaviors. Different load behaviors may belong to a same type. For example, both "Resource pool" and a "Priority" belong to "Workload management". The "Behavior" indicates a load behavior associated with a query_band key-value pair. The "Behavior description" describes a specific load behavior. The "Order" in the "Type" is used to indicate the priority of the query_band load behavior identification. When a session has multiple query_band key-value pairs, the query_band key-value pair with a smaller order value is preferentially used to identify a load behavior. Each query_band key-value pair can have multiple associated load behaviors, while one load behavior can only have one associated key-value pair. The query_band load behavior is described as follows:

- Resource pool: query_band can be associated with resource pools. During job execution, if a resource pool is associated with query_band, the resource pool is used in preference. Otherwise, the resource pool associated with the user is used.
  - When query_band is associated with a resource pool, an error is reported if the resource pool does not exist, and the association fails.
  - When query_band is associated with a resource pool, the dependency between query_band and the resource pool is recorded.
  - When a resource pool associated with query_band is deleted, a message is displayed indicating that the resource pool fails to be deleted because of the dependency between query_band and the resource pool.
- Intra-queue priority: query_band can be associated with job priorities, including high, medium, and low. Rush is provided as a special priority (green channel). The default priority is medium. In practice, most jobs use the medium priority, low-priority jobs use the low priority, and privileged jobs use the high priority. It is not recommended that a large number of jobs use the high priority. The rush priority is used only in special scenarios and is not recommended in normal cases.

  The intra-queue priority is used to implement the queuing priority.
  - In the static load management scenario, when the CN concurrency is insufficient, CN global queuing is triggered. The CN global queue is a priority queue.
  - In the dynamic load management scenario, if the DN memory is insufficient, CCN global queuing is triggered. The CCN global queue is a priority queue.
  - When the resource pool concurrency or memory is insufficient, resource pool queuing is triggered. The resource pool queue is a priority queue.

The preceding priority queues comply with the following scheduling rules:

- Jobs with a higher priority are scheduled first.
- After all jobs with a high priority are scheduled, jobs with a low priority are scheduled.
- In dynamic load management scenarios, the CN global queue does not support the query_band priority.

● Order: The identification order of query_bands can be configured. The default order value is **-1**. Except the default order value, there are no two query_bands with the same order value. The query_band order is verified when being configured. If there are query_bands with the same order value, the order values are recursively increased by 1 until there are no query_bands with the same order value.

  - If a session has multiple query_band key-value pairs, the query_band key-value pair with a smaller order value is used for load identification.
  - **0** is the smallest order value, and the default order value **-1** is the largest order value.
  - If the query_bands are all of the same order value, the anterior query_band is used for load identification.
  - For example, if in **set query_band='b=1;a=3;c=1'; b=1**, the order value of **b=1** is **-1**, **a=3** is **4**, **c=1** is **1**, **c=1** is used as the query_band for load identification. This design enables load administrators to adjust load scheduling.

Application and Configuration of query_band

● The **pg_workload_action** cross-database system catalog is used to store the query_band action and order. For details, see .

● The default action and order are not stored in the **pg_workload_action** system catalog. If a non-default action is set for query_band, the default action is also displayed when actions are queried. The message <query_band information not found> is displayed when the action and order to be queried are the default query_band action.

● The **gs_wlm_set_queryband_action** function sets the query_band sequence. The maximum length of the first parameter, that is, the query_band key value pair, is 63 characters. For the second parameter, it is case insensitive and multiple actions are separated by semicolons (;). **order** is the default parameter and its default value is **-1**. For details, see the function in section .

● The **gs_wlm_set_queryband_order** function sets the query_band sequence. The maximum length of the first parameter, that is, a query_band key value pair, is 63 characters. The value of query_band must be greater than or equal to **–1**. Except the default value **–1**, the value of query_band order must be unique. When setting the query_band order, if there are query_bands with the same order values, the original order value is increased by 1. For details, see the function in section .

● The **gs_wlm_get_queryband_action** function is used to query the query_band action. For details, see in section .

● **pg_queryband_action** provides the system view for querying all query_band actions. For details, see .

● The query_band priority is displayed as an integer in the load management view (). The mapping between numbers and priorities is as follows:

  - 0: not controlled by load management
  - 1: low
  - 2: medium
  - 4: high
  - 8: rush
- Permission control: Except initial users, other users have the permission to set and query query_band only when they are authorized.

NOTE

When all running jobs are canceled in batches or the maximum number of concurrent jobs in a queue is 1 and only one queue is running jobs, the CN may be triggered to automatically wake up jobs. As a result, jobs are not delivered by priority.

Examples

Step 1 Set the associated resource pool to **p1**, priority to **rush**, and order to **1** for query_band **JobName to abc**.

```
SELECT * FROM gs_wlm_set_queryband_action('JobName=abc','respool=p1;priority=rush',1);
gs_wlm_set_queryband_action
-----------------------------
 t
(1 row)
```

Step 2 Change the associated resource pool to **p2** for query_band **JobName=abc**.

```
SELECT * FROM gs_wlm_set_queryband_action('JobName=abc','respool=p2');
gs_wlm_set_queryband_action
-----------------------------
 t
(1 row)
```

Step 3 Change the priority to **high** for query_band **JobName=abc**.

```
SELECT * FROM gs_wlm_set_queryband_action('JobName=abc','priority=high');
gs_wlm_set_queryband_action
-----------------------------
 t
(1 row)
```

Step 4 Change the order to **3** for query_band **JobName=abc**.

```
SELECT * FROM gs_wlm_set_queryband_order('JobName=abc',3);
gs_wlm_set_queryband_order
-----------------------------
 t
(1 row)
```

Step 5 Query the load behaviors associated with query_band.

```
SELECT * FROM pg_queryband_action;
    qband     | respool_id | respool | priority | qborder
--------------+------------+---------+----------+---------
 AppName=test |      16974 | p1      | low      |      -1
 JobName=abc  |      17119 | p2      | high     |       1
(2 rows)
```

----End

13.5 SQL Tuning Examples

13.5.1 Case: Selecting an Appropriate Distribution Column

Distribution columns are used to distribute data to different nodes. A proper distribution key can avoid data skew.

When performing join query, you are advised to select the join condition in the query as the distribution key. When a join condition is used as a distribution key, related data is distributed locally on DNs, reducing the cost of data flow between DNs and improving the query speed.

Before optimization

Use **a** as the distribution column of **t1** and **t2**. The table definition is as follows:

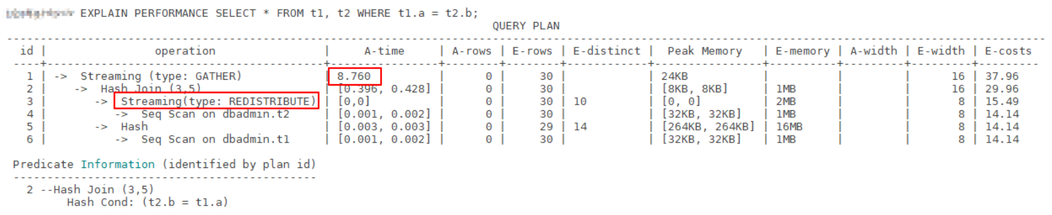
```
CREATE TABLE t1 (a int, b int) DISTRIBUTE BY HASH (a);
CREATE TABLE t2 (a int, b int) DISTRIBUTE BY HASH (a);
```

The following query is executed:

```
SELECT * FROM t1, t2 WHERE t1.a = t2.b;
```

In this case, the execution plan contains **Streaming(type: REDISTRIBUTE)**, that is, the DN redistributes data to all DNs based on the selected column. This will cause a large amount of data to be transmitted between DNs, as shown in **Figure 13-19**.

Figure 13-19 Selecting an appropriate distribution column (1)



```
[redacted] EXPLAIN PERFORMANCE SELECT * FROM t1, t2 WHERE t1.a = t2.b;
                                                             QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------------------------------------------
 id |                operation                 |      A-time      | A-rows | E-rows | E-distinct |  Peak Memory   | E-memory | A-width | E-width | E-costs
----+------------------------------------------+------------------+--------+--------+------------+----------------+----------+---------+---------+---------
  1 | ->  Streaming (type: GATHER)             | 8.760            |      0 |     30 |            | 24KB           |          |         |      16 | 37.96
  2 |    ->  Hash Join (3,5)                   | [0.396, 0.428]   |      0 |     30 |            | [8KB, 8KB]     | 1MB      |         |      16 | 29.96
  3 |       ->  Streaming(type: REDISTRIBUTE)  | [0,0]            |      0 |     30 | 10         | [0, 0]         | 2MB      |         |       8 | 15.49
  4 |          ->  Seq Scan on dbadmin.t2      | [0.001, 0.002]   |      0 |     30 |            | [32KB, 32KB]   | 1MB      |         |       8 | 14.14
  5 |       ->  Hash                           | [0.003, 0.003]   |      0 |     29 | 14         | [264KB, 264KB] | 16MB     |         |       8 | 14.14
  6 |          ->  Seq Scan on dbadmin.t1      | [0.001, 0.002]   |      0 |     30 |            | [32KB, 32KB]   | 1MB      |         |       8 | 14.14

 Predicate Information (identified by plan id)
 ---------------------------------------------
   2 --Hash Join (3,5)
         Hash Cond: (t2.b = t1.a)
```

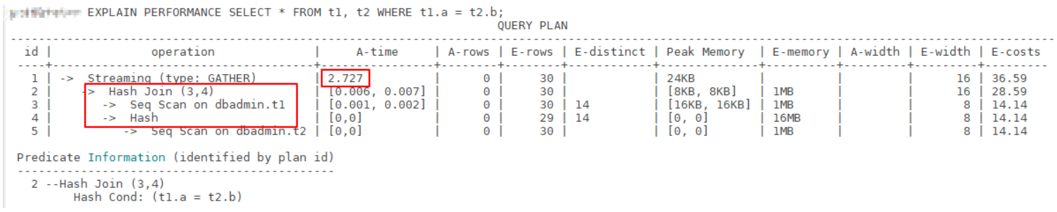
After optimization

Use the join condition in the query as the distribution key and run the following statement to changethe distribution key of **t2** as **b**:

```
ALTER TABLE t2 DISTRIBUTE BY HASH (b);
```

After the distribution column of table **t2** is changed to column **b**, the execution plan does not contain **Streaming(type: REDISTRIBUTE)**. This reduces the amount of communication data between DNs and reduces the execution time from 8.7 ms to 2.7 ms, improving query performance, as shown in **Figure 13-20**.

Figure 13-20 Selecting an appropriate distribution column (2)



```
[redacted] EXPLAIN PERFORMANCE SELECT * FROM t1, t2 WHERE t1.a = t2.b;
                                                             QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------------------------------------------
 id |                operation                |      A-time      | A-rows | E-rows | E-distinct |  Peak Memory   | E-memory | A-width | E-width | E-costs
----+-----------------------------------------+------------------+--------+--------+------------+----------------+----------+---------+---------+---------
  1 | ->  Streaming (type: GATHER)            | 2.727            |      0 |     30 |            | 24KB           |          |         |      16 | 36.59
  2 |    ->  Hash Join (3,4)                  | [0.006, 0.007]   |      0 |     30 |            | [8KB, 8KB]     | 1MB      |         |      16 | 28.59
  3 |       ->  Seq Scan on dbadmin.t1        | [0.001, 0.002]   |      0 |     30 | 14         | [16KB, 16KB]   | 1MB      |         |       8 | 14.14
  4 |       ->  Hash                          | [0,0]            |      0 |     29 | 14         | [0, 0]         | 16MB     |         |       8 | 14.14
  5 |          ->  Seq Scan on dbadmin.t2     | [0,0]            |      0 |     30 |            | [0, 0]         | 1MB      |         |       8 | 14.14

 Predicate Information (identified by plan id)
 ---------------------------------------------
   2 --Hash Join (3,4)
         Hash Cond: (t1.a = t2.b)
```

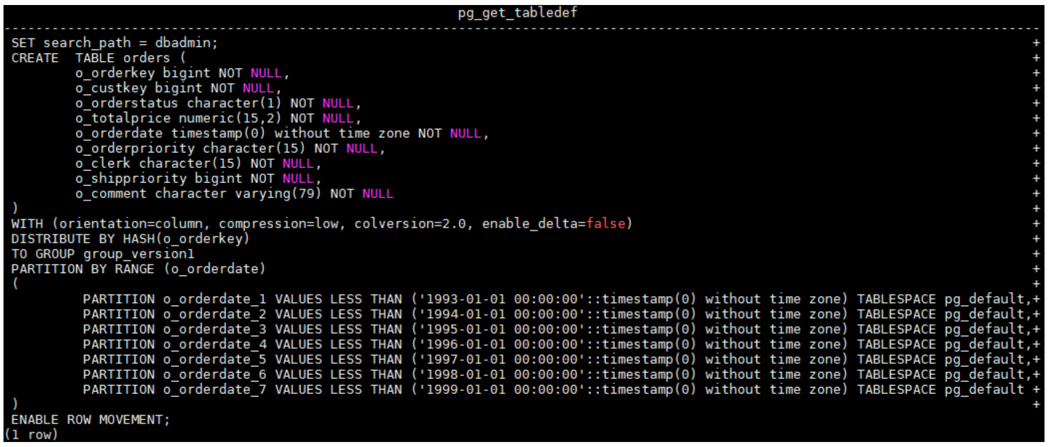
13.5.2 Case: Creating an Appropriate Index

Creating a proper index can accelerate the retrieval of data rows in a table. Indexes occupy disk space and reduce the speed of adding, deleting, and updating rows. If data needs to be updated very frequently or disk space is limited, you need to limit the number of indexes. Create indexes for large tables. Because the more data in the table, the more effective the index is. You are advised to create indexes on:

- Columns that need to be queried frequently
- Joined columns. For a query on joined columns, you are advised to create a composite index on the joined columns. For example, if the join condition is **select * from t1 join t2 on t1.a=t2.a and t1.b=t2.b**. You can create a composite index on the **a** and **b** columns of table **t1**.
- Columns having filter criteria (especially scope criteria) of a **where** clause
- Columns that appear after **order by**, **group by**, and **distinct**

Before optimization

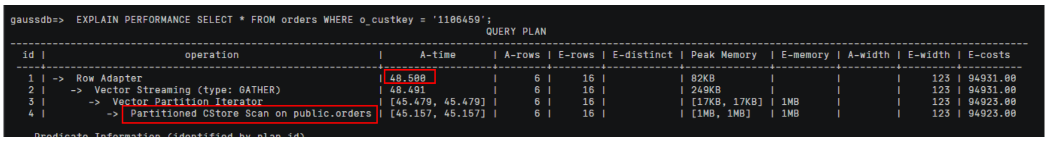
The column-store partitioned table **orders** is defined as follows:



```
                                                        pg_get_tabledef
------------------------------------------------------------------------------------------------------------------------------------
 SET search_path = dbadmin;                                                                                                        +
 CREATE  TABLE orders (                                                                                                            +
         o_orderkey bigint NOT NULL,                                                                                               +
         o_custkey bigint NOT NULL,                                                                                                +
         o_orderstatus character(1) NOT NULL,                                                                                      +
         o_totalprice numeric(15,2) NOT NULL,                                                                                      +
         o_orderdate timestamp(0) without time zone NOT NULL,                                                                      +
         o_orderpriority character(15) NOT NULL,                                                                                   +
         o_clerk character(15) NOT NULL,                                                                                           +
         o_shippriority bigint NOT NULL,                                                                                           +
         o_comment character varying(79) NOT NULL                                                                                  +
 )                                                                                                                                 +
 WITH (orientation=column, compression=low, colversion=2.0, enable_delta=false)                                                    +
 DISTRIBUTE BY HASH(o_orderkey)                                                                                                    +
 TO GROUP group_version1                                                                                                           +
 PARTITION BY RANGE (o_orderdate)                                                                                                  +
 (                                                                                                                                 +
          PARTITION o_orderdate_1 VALUES LESS THAN ('1993-01-01 00:00:00'::timestamp(0) without time zone) TABLESPACE pg_default,+
          PARTITION o_orderdate_2 VALUES LESS THAN ('1994-01-01 00:00:00'::timestamp(0) without time zone) TABLESPACE pg_default,+
          PARTITION o_orderdate_3 VALUES LESS THAN ('1995-01-01 00:00:00'::timestamp(0) without time zone) TABLESPACE pg_default,+
          PARTITION o_orderdate_4 VALUES LESS THAN ('1996-01-01 00:00:00'::timestamp(0) without time zone) TABLESPACE pg_default,+
          PARTITION o_orderdate_5 VALUES LESS THAN ('1997-01-01 00:00:00'::timestamp(0) without time zone) TABLESPACE pg_default,+
          PARTITION o_orderdate_6 VALUES LESS THAN ('1998-01-01 00:00:00'::timestamp(0) without time zone) TABLESPACE pg_default,+
          PARTITION o_orderdate_7 VALUES LESS THAN ('1999-01-01 00:00:00'::timestamp(0) without time zone) TABLESPACE pg_default +
 )                                                                                                                                 +
 ENABLE ROW MOVEMENT;
(1 row)
```

Run the SQL statement to query the execution plan when no index is created. It is found that the execution time is 48 milliseconds.

```
EXPLAIN PERFORMANCE SELECT * FROM orders WHERE o_custkey = '1106459';
```



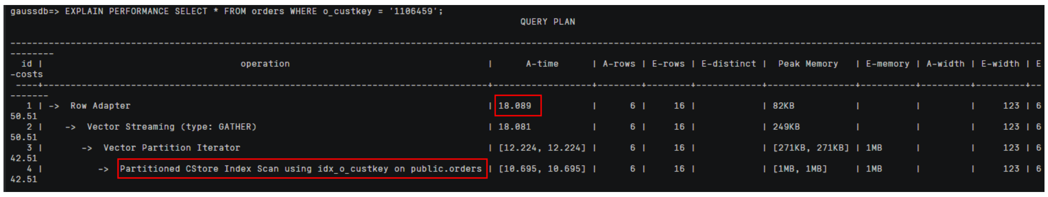
```
gaussdb=>  EXPLAIN PERFORMANCE SELECT * FROM orders WHERE o_custkey = '1106459';
                                                                 QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------------------------
 id |                     operation                     |       A-time       | A-rows | E-rows | E-distinct | Peak Memory  | E-memory | A-width | E-width | E-costs
----+---------------------------------------------------+--------------------+--------+--------+------------+--------------+----------+---------+---------+----------
  1 | ->  Row Adapter                                   | 48.500             |      6 |     16 |            | 82KB         |          |         |     123 | 94931.00
  2 |    ->  Vector Streaming (type: GATHER)            | 48.491             |      6 |     16 |            | 249KB        |          |         |     123 | 94931.00
  3 |       ->  Vector Partition Iterator               | [45.479, 45.479] |      6 |     16 |            | [17KB, 17KB] | 1MB      |         |     123 | 94923.00
  4 |          ->  Partitioned CStore Scan on public.orders | [45.157, 45.157] |      6 |     16 |            | [1MB, 1MB]   | 1MB      |         |     123 | 94923.00

   Predicate Information (identified by plan_id)
```

After optimization

The filtering condition column of the **where** clause is **o_custkey**. Add an index to the **o_custkey** column.

```
CREATE INDEX idx_o_custkey ON orders (o_custkey) LOCAL;
```

Run the SQL statement to query the execution plan after the index is created. It is found that the execution time is 18 milliseconds.



```
gaussdb=> EXPLAIN PERFORMANCE SELECT * FROM orders WHERE o_custkey = '1106459';
                                                                                    QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------
 id |                           operation                            |      A-time       | A-rows | E-rows | E-distinct |  Peak Memory    | E-memory | A-width | E-width | E
-costs
 ----+-----------------------------------------------------------------+-------------------+--------+--------+------------+-----------------+----------+---------+---------+--
-------
   1 | ->  Row Adapter                                                  | 18.089            |      6 |     16 |            | 82KB            |          |         |     123 | 6
50.51
   2 |    ->  Vector Streaming (type: GATHER)                           | 18.081            |      6 |     16 |            | 249KB           |          |         |     123 | 6
50.51
   3 |       ->  Vector Partition Iterator                              | [12.224, 12.224] |      6 |     16 |            | [271KB, 271KB] | 1MB      |         |     123 | 6
42.51
   4 |          ->  Partitioned CStore Index Scan using idx_o_custkey on public.orders | [10.695, 10.695] |      6 |     16 |            | [1MB, 1MB]     | 1MB      |         |     123 | 6
42.51
```

13.5.3 Case: Adding NOT NULL for JOIN Columns

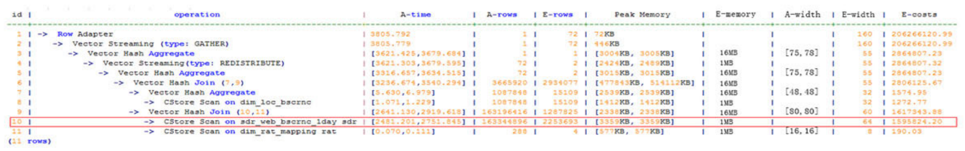
If there are many **NULL** values in the **JOIN** columns, you can add the filter criterion **IS NOT NULL** to filter data in advance to improve the **JOIN** efficiency.

Before optimization

```
SELECT
 *
FROM
( ( SELECT
  STARTTIME STTIME,
  SUM(NVL(PAGE_DELAY_MSEL,0)) PAGE_DELAY_MSEL,
  SUM(NVL(PAGE_SUCCEED_TIMES,0)) PAGE_SUCCEED_TIMES,
  SUM(NVL(FST_PAGE_REQ_NUM,0)) FST_PAGE_REQ_NUM,
  SUM(NVL(PAGE_AVG_SIZE,0)) PAGE_AVG_SIZE,
  SUM(NVL(FST_PAGE_ACK_NUM,0)) FST_PAGE_ACK_NUM,
  SUM(NVL(DATATRANS_DW_DURATION,0)) DATATRANS_DW_DURATION,
  SUM(NVL(PAGE_SR_DELAY_MSEL,0)) PAGE_SR_DELAY_MSEL
 FROM
  PS.SDR_WEB_BSCRNC_1DAY SDR
  INNER JOIN (SELECT
      BSCRNC_ID,
      BSCRNC_NAME,
      ACCESS_TYPE,
      ACCESS_TYPE_ID
     FROM
      nethouse.DIM_LOC_BSCRNC
     GROUP BY
      BSCRNC_ID,
      BSCRNC_NAME,
      ACCESS_TYPE,
      ACCESS_TYPE_ID) DIM
  ON SDR.BSCRNC_ID = DIM.BSCRNC_ID
  AND DIM.ACCESS_TYPE_ID IN (0,1,2)
  INNER JOIN nethouse.DIM_RAT_MAPPING RAT
  ON (RAT.RAT = SDR.RAT)
 WHERE
  ( (STARTTIME >= 1461340800
  AND STARTTIME < 1461427200) )
  AND RAT.ACCESS_TYPE_ID IN (0,1,2)
 GROUP BY STTIME ) ) ;
```

Figure 13-21 shows the execution plan.

Figure 13-21 Adding NOT NULL for JOIN columns (1)



```
 id |                    operation                          |        A-time         |  A-rows   |  E-rows  |      Peak Memory        | E-memory |  A-width | E-width |   E-costs
----+-------------------------------------------------------+-----------------------+-----------+----------+-------------------------+----------+----------+---------+--------------
  1 | ->  Row Adapter                                       | 3805.792              |         1 |       72 | 72KB                    |          |          |     160 | 206266120.99
  2 |    ->  Vector Streaming (type: GATHER)                | 3805.779              |         1 |       72 | 446KB                   |          |          |     160 | 206266120.99
  3 |       ->  Vector Hash Aggregate                       | [3621.425,3679.684]   |         1 |        1 | [3004KB, 3005KB]        | 16MB     | [75, 78] |      55 | 2864807.23
  4 |          ->  Vector Streaming(type: REDISTRIBUTE)     | [3621.303,3679.595]   |        72 |        2 | [2424KB, 2489KB]        | 1MB      |          |      55 | 2864807.32
  5 |             ->  Vector Hash Aggregate                 | [3316.657,3634.515]   |        72 |        2 | [3015KB, 3015KB]        | 16MB     | [75, 78] |      55 | 2864807.23
  6 |                ->  Vector Hash Join (7,9)             | [3236.674,3540.294]   |   3665920 |  2934077 | [477843KB, 514112KB]    | 16MB     |          |      55 | 2806125.67
  7 |                   ->  Vector Hash Aggregate           | [5.630,6.979]         |   1087848 |    15109 | [2539KB, 2539KB]        | 16MB     | [48, 48] |      32 | 1574.95
  8 |                      ->  CStore Scan on dim_loc_bscrnc | [1.071,1.229]        |   1087848 |    15109 | [1412KB, 1412KB]        | 1MB      |          |      32 | 1272.77
  9 |                   ->  Vector Hash Join (10,11)        | [2641.130,2919.618]   | 163196416 |  1287825 | [2338KB, 2338KB]        | 16MB     | [80, 80] |      60 | 1617343.88
 10 |                      ->  CStore Scan on sdr_web_bscrnc_1day sdr | [2481.201,2751.845] | 163344896 | 2253693 | [3359KB, 3359KB] | 1MB |     |      64 | 1595824.20
 11 |                      ->  CStore Scan on dim_rat_mapping rat | [0.070,0.111]     |       288 |        4 | [577KB, 577KB]          | 1MB      | [16, 16] |       8 | 190.03
(11 rows)
```

After optimization

1. As shown in **Figure 13-21**, the sequential scan phase is time consuming.

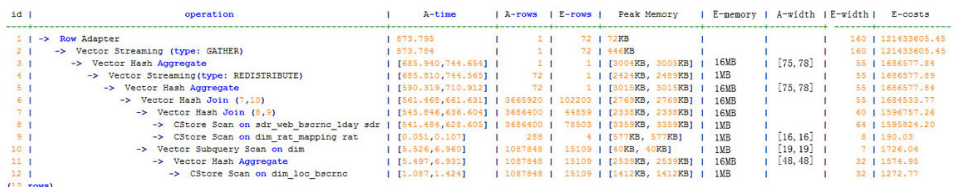
2. The JOIN performance is poor because a large number of null values exist in the JOIN column **BSCRNC_ID** of the PS.SDR_WEB_BSCRNC_1DAY table.

   Therefore, you are advised to manually add **NOT NULL** for **JOIN** columns in the statement, as shown below:

```
SELECT
 *
FROM
( ( SELECT
  STARTTIME STTIME,
  SUM(NVL(PAGE_DELAY_MSEL,0)) PAGE_DELAY_MSEL,
  SUM(NVL(PAGE_SUCCEED_TIMES,0)) PAGE_SUCCEED_TIMES,
  SUM(NVL(FST_PAGE_REQ_NUM,0)) FST_PAGE_REQ_NUM,
  SUM(NVL(PAGE_AVG_SIZE,0)) PAGE_AVG_SIZE,
  SUM(NVL(FST_PAGE_ACK_NUM,0)) FST_PAGE_ACK_NUM,
  SUM(NVL(DATATRANS_DW_DURATION,0)) DATATRANS_DW_DURATION,
  SUM(NVL(PAGE_SR_DELAY_MSEL,0)) PAGE_SR_DELAY_MSEL
 FROM
  PS.SDR_WEB_BSCRNC_1DAY SDR
  INNER JOIN (SELECT
      BSCRNC_ID,
      BSCRNC_NAME,
      ACCESS_TYPE,
      ACCESS_TYPE_ID
     FROM
      nethouse.DIM_LOC_BSCRNC
     GROUP BY
      BSCRNC_ID,
      BSCRNC_NAME,
      ACCESS_TYPE,
      ACCESS_TYPE_ID) DIM
  ON SDR.BSCRNC_ID = DIM.BSCRNC_ID
  AND DIM.ACCESS_TYPE_ID IN (0,1,2)
  INNER JOIN nethouse.DIM_RAT_MAPPING RAT
  ON (RAT.RAT = SDR.RAT)
 WHERE
  ( (STARTTIME >= 1461340800
  AND STARTTIME < 1461427200) )
  AND RAT.ACCESS_TYPE_ID IN (0,1,2)
  and SDR.BSCRNC_ID is not null
 GROUP BY
  STTIME ) ) A;
```

   Figure 13-22 shows the execution plan.

   Figure 13-22 Adding NOT NULL for JOIN columns (2)



```
 id |                          operation                          |       A-time        | A-rows  | E-rows |   Peak Memory      | E-memory | A-width |E-width|   E-costs
----+-------------------------------------------------------------+---------------------+---------+--------+--------------------+----------+---------+-------+---------------
  1 | ->  Row Adapter                                             | 873.795             |       1 |     72 | 72KB               |          |         |   160 | 121433605.45
  2 |   ->  Vector Streaming (type: GATHER)                       | 873.784             |       1 |     72 | 446KB              |          |         |   160 | 121433605.45
  3 |     ->  Vector Hash Aggregate                               | [685.940,744.654]   |       1 |      1 | [3004KB, 3005KB]   | 16MB     | [75,78] |    55 | 1686577.84
  4 |       ->  Vector Streaming(type: REDISTRIBUTE)              | [685.810,744.565]   |      72 |      1 | [2424KB, 2489KB]   | 1MB      |         |    55 | 1686577.89
  5 |         ->  Vector Hash Aggregate                           | [590.319,710.912]   |      72 |      1 | [3015KB, 3015KB]   | 16MB     | [75,78] |    55 | 1686577.84
  6 |           ->  Vector Hash Join (7,10)                       | [561.468,661.631]   | 3665920 | 102203 | [2769KB, 2769KB]   | 16MB     |         |    55 | 1684533.77
  7 |             ->  Vector Hash Join (8,9)                      | [545.846,636.604]   | 3686400 |  44859 | [2338KB, 2338KB]   | 16MB     |         |    60 | 1596757.26
  8 |               ->  CStore Scan on sdr_web_bscrnc_1day sdr    | [541.484,628.605]   | 3686400 |  78503 | [3359KB, 3359KB]   | 1MB      |         |    64 | 1595824.20
  9 |               ->  CStore Scan on dim_rat_mapping rat        | [0.051,0.107]       |     288 |      4 | [577KB, 577KB]     | 1MB      | [16,16] |     8 | 190.03
 10 |             ->  Vector Subquery Scan on dim                 | [5.526,6.960]       | 1087848 |  15109 | [40KB, 40KB]       | 1MB      | [19,19] |     7 | 1726.04
 11 |               ->  Vector Hash Aggregate                     | [5.497,6.931]       | 1087848 |  15109 | [2539KB, 2539KB]   | 16MB     | [48,48] |    32 | 1574.95
 12 |                 ->  CStore Scan on dim_loc_bscrnc           | [1.087,1.424]       | 1087848 |  15109 | [1412KB, 1412KB]   | 1MB      |         |    32 | 1272.77
(12 rows)
```

13.5.4 Case: Pushing Down Sort Operations to DNs

In an execution plan, more than 95% of the execution time is spent on **window agg** performed on the CN. In this case, **sum** is performed for the two columns separately, and then another **sum** is performed for the separate sum results of the two columns. After this, trunc and sorting are performed in sequence. You can try to rewrite the statement into a subquery to push down the sorting operations.

Before optimization

The table structure is as follows:

```
CREATE TABLE public.test(imsi int,L4_DW_THROUGHPUT int,L4_UL_THROUGHPUT int)
with (orientation = column) DISTRIBUTE BY hash(imsi);
```

The query statements are as follows:

```
SELECT COUNT(1) over() AS DATACNT,
IMSI AS IMSI_IMSI,
CAST(TRUNC(((SUM(L4_UL_THROUGHPUT) + SUM(L4_DW_THROUGHPUT))), 0) AS
DECIMAL(20)) AS TOTAL_VOLOME_KPIID
FROM public.test AS test
GROUP BY IMSI
ORDER BY TOTAL_VOLOME_KPIID DESC LIMIT 10;
```

The execution plan is as follows:

```
QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------
  id |                    operation                   |      A-time      | A-rows  | E-rows  | E-distinct | Peak Memory  |   E-
memory   | A-width | E-width | E-costs
 ----+--------------------------------------------------+------------------+---------+---------+------------+--------------
+--------------+---------+---------+----------
   1 | ->  Row Adapter                              | 2862.008         |      10 |      10 |            | 31KB         |
|        |      28 | 48360.42
   2 |    ->  Vector Limit                          | 2861.969         |      10 |      10 |            | 8KB          |
|        |      28 | 48360.42
   3 |      ->  Vector Sort                         | 2861.946         |      10 | 1000000 |            | 479KB
|            |        |      28 | 50860.39
   4 |         ->  Vector WindowAgg                  | 2166.759         | 1000000 | 1000000 |            | 69987KB
|            |        |      28 | 26750.75
   5 |            ->  Vector Streaming (type: GATHER)  | 136.813          | 1000000 | 1000000 |            |
208KB        |             |        |      28 | 15500.75
   6 |               ->  Vector Sonic Hash Aggregate   | [71.374, 73.640] | 1000000 | 1000000 |            | [14MB,
14MB] | 96MB(2919MB) | [31,31] |      28 | 15032.00
   7 |                  ->  CStore Scan on public.test | [2.957, 2.994]   | 1000000 | 1000000 |            | [1MB,
1MB]   | 1MB          |        |      12 | 1282.00
```

As we can see, both **window agg** and **sort** are performed on the CN, which is time consuming.

After optimization

Modify the statement to a subquery statement, as shown below:

```
SELECT COUNT(1) over() AS DATACNT, IMSI_IMSI, TOTAL_VOLOME_KPIID
FROM (SELECT IMSI AS IMSI_IMSI,
CAST(TRUNC(((SUM(L4_UL_THROUGHPUT) + SUM(L4_DW_THROUGHPUT))),
0) AS DECIMAL(20)) AS TOTAL_VOLOME_KPIID
FROM public.test AS test
GROUP BY IMSI
ORDER BY TOTAL_VOLOME_KPIID DESC LIMIT 10);
```

Perform **sum** on the **trunc** results of the two columns, take it as a subquery, and then perform **window agg** for the subquery to push down the sorting operation to DNs, as shown below:

```
 QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------
  id |                        operation                        |       A-time       | A-rows  | E-rows  | E-distinct |  Peak
Memory    |    E-memory    | A-width | E-width | E-costs
 ----+---------------------------------------------------------+--------------------+---------+---------+------------
+----------------+--------------+---------+---------+----------
   1 | ->  Row Adapter                                       | 955.277            |      10 |       5 |            | 31KB
```

```
|            |         |      24 | 25843.13
   2 |    ->  Vector WindowAgg                          | 955.261            |      10 |       5 |            | 1572KB
|            |         |      24 | 25843.13
   3 |      ->  Vector Streaming (type: GATHER)             | 955.015            |      10 |      10 |            |
127KB          |              |         |      24 | 25843.07
   4 |         ->  Vector Limit                           | [0.018, 0.018]     |      10 |      10 |            | [8KB, 8KB]     |
1MB          |         |      28 | 25836.97
   5 |           ->  Vector Streaming(type: BROADCAST)      | [0.014, 0.014]     |      20 |      20 |            |
[719KB, 719KB] | 2MB          |         |      28 | 25837.12
   6 |              ->  Vector Limit                       | [927.730, 934.283] |      20 |      20 |            | [8KB, 8KB]
| 1MB          |         |      28 | 25836.85
   7 |                ->  Vector Sort                      | [927.720, 934.269] |      20 | 1000000 |            | [463KB,
463KB] | 16MB         | [32,32] |      28 | 27086.82
   8 |                  ->  Vector Sonic Hash Aggregate   | [456.841, 461.077] | 1000000 | 1000000 |            |
[15MB, 15MB]   | 96MB(2916MB) | [31,31] |      28 | 15032.00
   9 |                     ->  CStore Scan on public.test | [2.959, 3.014]     | 1000000 | 1000000 |            | [1MB,
1MB]     | 1MB          |         |      12 | 1282.00
```

The optimized SQL statement greatly improves the performance by reducing the execution time from 2.862s to 0.955s. Note that the optimization result in this example is for reference only. Due to the uncertainty of **WindowAgg**, the optimized result set is related to the actual service.

13.5.5 Case: Configuring cost_param for Better Query Performance

The cost_param parameter is used to control use of different estimation methods in specific customer scenarios, allowing estimated values to be close to onsite values. This parameter can control various methods simultaneously by performing AND (&) operations on the bit for each method. A method is selected if its value is not **0**.

Scenario 1: Before Optimization

If **bit0** of **cost_param** is set to **1**, an improved mechanism is used for estimating the selection rate of non-equi-joins. This method is more accurate for estimating the selection rate of joins between two identical tables. The following example describes the optimization scenario when **bit0** of **cost_param** is set to **1**. In V300R002C00 and later, **cost_param & 1=0** is not used. That is, an optimized formula is selected for calculation.

NOTE

The selection rate indicates the percentage for which the number of rows meeting the join conditions account of the **JOIN** results when the **JOIN** relationship is established between two tables.

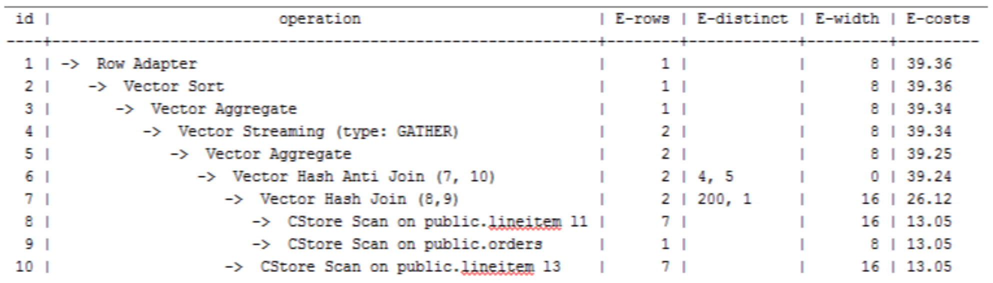
The table structure is as follows:

```
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) distribute by hash(L_ORDERKEY);

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) distribute by hash(O_ORDERKEY);
```

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;
```

The following figure shows the execution plan. (When **verbose** is used, **distinct** is added for column selection which is controlled by **cost off/on**. The hash join rows show the estimated number of distinct values and the other rows do not.)

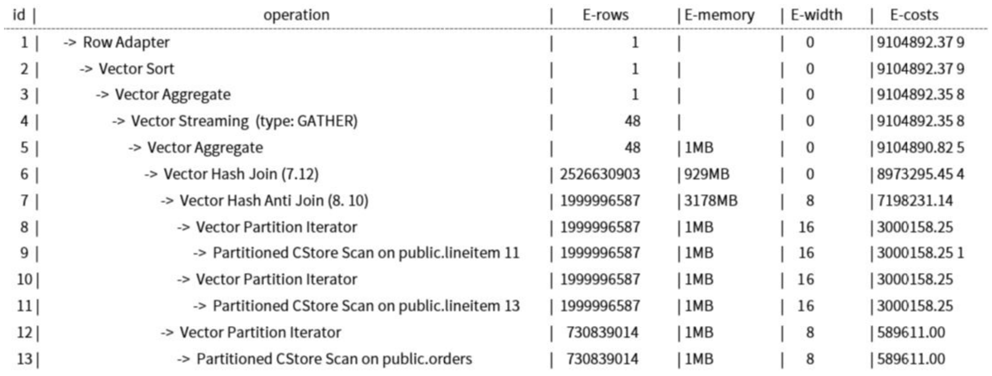


```
 id |                          operation                          | E-rows | E-distinct | E-width | E-costs
----+-------------------------------------------------------------+--------+------------+---------+---------
  1 | ->  Row Adapter                                             |      1 |            |       8 | 39.36
  2 |    ->  Vector Sort                                          |      1 |            |       8 | 39.36
  3 |       ->  Vector Aggregate                                  |      1 |            |       8 | 39.34
  4 |          ->  Vector Streaming (type: GATHER)                |      2 |            |       8 | 39.34
  5 |             ->  Vector Aggregate                            |      2 |            |       8 | 39.25
  6 |                ->  Vector Hash Anti Join (7, 10)            |      2 | 4, 5       |       0 | 39.24
  7 |                   ->  Vector Hash Join (8,9)                |      2 | 200, 1     |      16 | 26.12
  8 |                      ->  CStore Scan on public.lineitem l1 |      7 |            |      16 | 13.05
  9 |                      ->  CStore Scan on public.orders       |      1 |            |       8 | 13.05
 10 |                   ->  CStore Scan on public.lineitem l3     |      7 |            |      16 | 13.05
```

Scenario 1: After Optimization

These queries are from Anti Join connected in the **lineitem** table. When **cost_param & bit0** is **0**, the estimated number of Anti Join rows greatly differs from that of the actual number of rows, compromising the query performance. You can estimate the number of Anti Join rows more accurately by setting

cost_param & bit0 to **1** to improve the query performance. The optimized execution plan is as follows:



```
 id |                         operation                          |   E-rows    | E-memory  | E-width |   E-costs
----------------------------------------------------------------------------------------------------------------------
  1 |   -> Row Adapter                                           |           1 |           |    0    | 9104892.37 9
  2 |     -> Vector Sort                                         |           1 |           |    0    | 9104892.37 9
  3 |       -> Vector Aggregate                                  |           1 |           |    0    | 9104892.35 8
  4 |         -> Vector Streaming  (type: GATHER)                |          48 |           |    0    | 9104892.35 8
  5 |           -> Vector Aggregate                              |          48 | 1MB       |    0    | 9104890.82 5
  6 |             -> Vector Hash Join (7.12)                     | 2526630903  | 929MB     |    0    | 8973295.45 4
  7 |               -> Vector Hash Anti Join (8. 10)             | 1999996587  | 3178MB    |    8    | 7198231.14
  8 |                 -> Vector Partition Iterator               | 1999996587  | 1MB       |   16    | 3000158.25
  9 |                   -> Partitioned CStore Scan on public.lineitem 11 | 1999996587 | 1MB |   16    | 3000158.25 1
 10 |                 -> Vector Partition Iterator               | 1999996587  | 1MB       |   16    | 3000158.25
 11 |                   -> Partitioned CStore Scan on public.lineitem 13 | 1999996587 | 1MB |   16    | 3000158.25
 12 |               -> Vector Partition Iterator                 |  730839014  | 1MB       |    8    | 589611.00
 13 |                 -> Partitioned CStore Scan on public.orders |  730839014  | 1MB      |    8    | 589611.00
```

Scenario 2: Before Optimization

If **bit1** is set to **1** (**set cost_param=2**), 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. The following example describes the optimization scenario when **bit1** of **cost_param** is set to **1**.

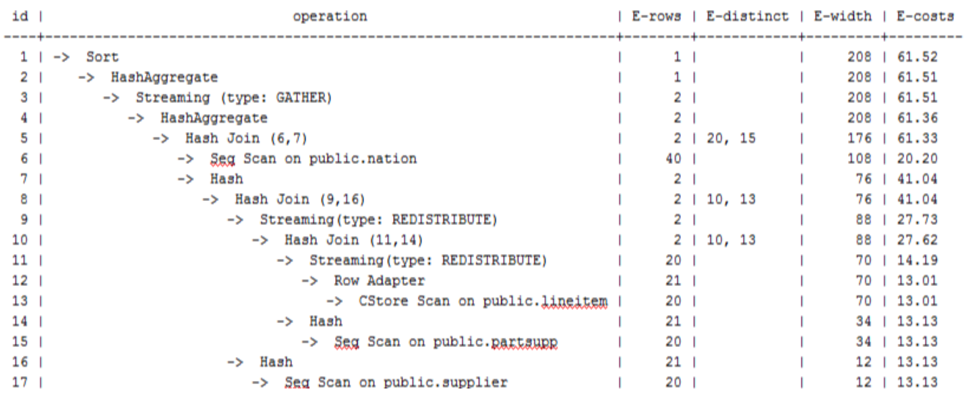
The table structure is as follows:

```
CREATE TABLE NATION
(
N_NATIONKEYINT NOT NULL
, N_NAMECHAR(25) NOT NULL
, N_REGIONKEYINT NOT NULL
, N_COMMENTVARCHAR(152)
) distribute by replication;
CREATE TABLE SUPPLIER
(
S_SUPPKEYBIGINT NOT NULL
, S_NAMECHAR(25) NOT NULL
, S_ADDRESSVARCHAR(40) NOT NULL
, S_NATIONKEYINT NOT NULL
, S_PHONECHAR(15) NOT NULL
, S_ACCTBALDECIMAL(15,2) NOT NULL
, S_COMMENTVARCHAR(101) NOT NULL
) distribute by hash(S_SUPPKEY);
CREATE TABLE PARTSUPP
(
PS_PARTKEYBIGINT NOT NULL
, PS_SUPPKEYBIGINT NOT NULL
, PS_AVAILQTYBIGINT NOT NULL
, PS_SUPPLYCOSTDECIMAL(15,2)NOT NULL
, PS_COMMENTVARCHAR(199) NOT NULL
)distribute by hash(PS_PARTKEY);
```

The query statements are as follows:

```
set cost_param=2;
explain verbose select
nation,
sum(amount) as sum_profit
from
(
select
n_name as nation,
l_extendedprice * (1 - l_discount) - ps_supplycost * l_quantity as amount
from
supplier,
lineitem,
partsupp,
nation
where
s_suppkey = l_suppkey
and ps_suppkey = l_suppkey
and ps_partkey = l_partkey
and s_nationkey = n_nationkey
) as profit
group by nation
order by nation;
```

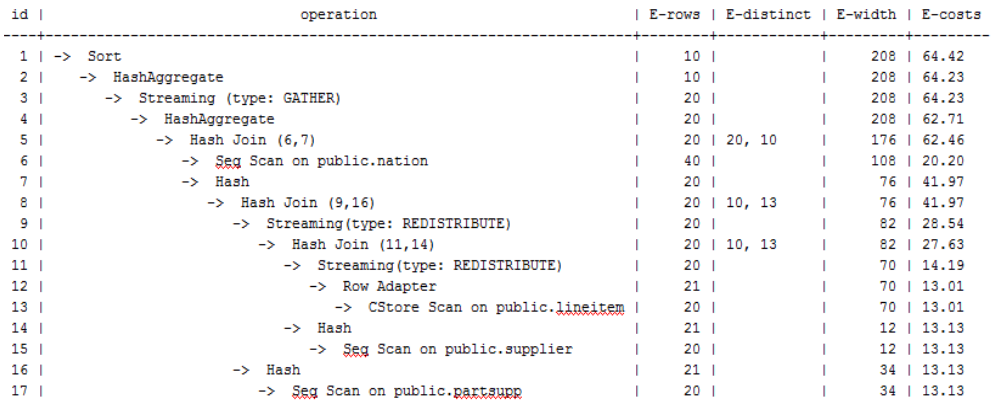
When **bit1** of **cost_param** is **0**, the execution plan is shown as follows:



```
 id |                          operation                              | E-rows | E-distinct | E-width | E-costs
----+-----------------------------------------------------------------+--------+------------+---------+---------
  1 | ->  Sort                                                        |      1 |            |     208 | 61.52
  2 |    ->  HashAggregate                                            |      1 |            |     208 | 61.51
  3 |       ->  Streaming (type: GATHER)                              |      2 |            |     208 | 61.51
  4 |          ->  HashAggregate                                      |      2 |            |     208 | 61.36
  5 |             ->  Hash Join (6,7)                                 |      2 | 20, 15     |     176 | 61.33
  6 |                ->  Seq Scan on public.nation                    |     40 |            |     108 | 20.20
  7 |                ->  Hash                                         |      2 |            |      76 | 41.04
  8 |                   ->  Hash Join (9,16)                          |      2 | 10, 13     |      76 | 41.04
  9 |                      ->  Streaming(type: REDISTRIBUTE)          |      2 |            |      88 | 27.73
 10 |                         ->  Hash Join (11,14)                  |      2 | 10, 13     |      88 | 27.62
 11 |                            ->  Streaming(type: REDISTRIBUTE)   |     20 |            |      70 | 14.19
 12 |                               ->  Row Adapter                  |     21 |            |      70 | 13.01
 13 |                                  ->  CStore Scan on public.lineitem |     20 |            |      70 | 13.01
 14 |                            ->  Hash                            |     21 |            |      34 | 13.13
 15 |                               ->  Seq Scan on public.partsupp  |     20 |            |      34 | 13.13
 16 |                      ->  Hash                                  |     21 |            |      12 | 13.13
 17 |                         ->  Seq Scan on public.supplier        |     20 |            |      12 | 13.13
```

Scenario 2: After Optimization

In the preceding queries, the hash join criteria of the supplier, lineitem, and partsupp tables are setting **lineitem.l_suppkey** to **supplier.s_suppkey** and **lineitem.l_partkey** to **partsupp.ps_partkey**. Two filter criteria exist in the hash join conditions. **lineitem.l_suppkey** in the first filter criteria and **lineitem.l_partkey** in the second filter criteria are two columns with strong relationship of the lineitem table. In this situation, when you estimate the rate of the hash join conditions, if **cost_param & bit1** is **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. The plan after optimization is shown as follows:



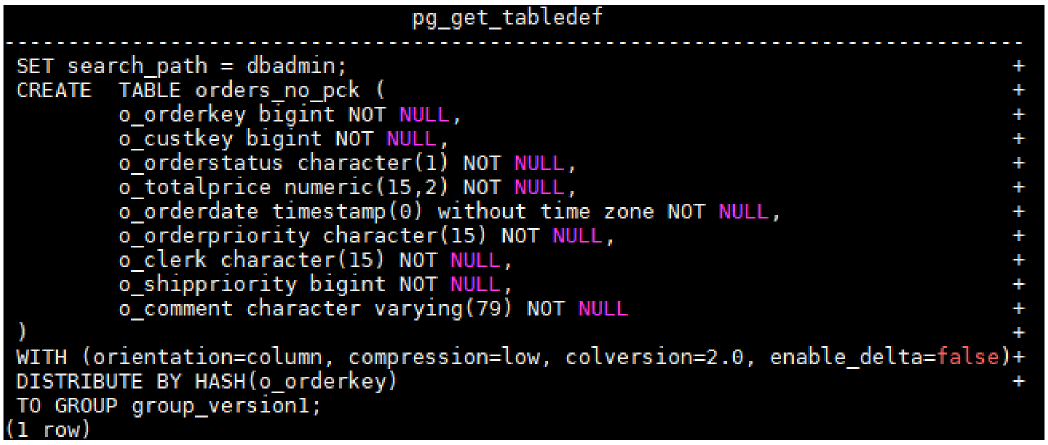
```
 id |                          operation                              | E-rows | E-distinct | E-width | E-costs
----+-----------------------------------------------------------------+--------+------------+---------+---------
  1 | ->  Sort                                                        |     10 |            |     208 | 64.42
  2 |    ->  HashAggregate                                            |     10 |            |     208 | 64.23
  3 |       ->  Streaming (type: GATHER)                              |     20 |            |     208 | 64.23
  4 |          ->  HashAggregate                                      |     20 |            |     208 | 62.71
  5 |             ->  Hash Join (6,7)                                 |     20 | 20, 10     |     176 | 62.46
  6 |                ->  Seq Scan on public.nation                    |     40 |            |     108 | 20.20
  7 |                ->  Hash                                         |     20 |            |      76 | 41.97
  8 |                   ->  Hash Join (9,16)                          |     20 | 10, 13     |      76 | 41.97
  9 |                      ->  Streaming(type: REDISTRIBUTE)          |     20 |            |      82 | 28.54
 10 |                         ->  Hash Join (11,14)                  |     20 | 10, 13     |      82 | 27.63
 11 |                            ->  Streaming(type: REDISTRIBUTE)   |     20 |            |      70 | 14.19
 12 |                               ->  Row Adapter                  |     21 |            |      70 | 13.01
 13 |                                  ->  CStore Scan on public.lineitem |     20 |            |      70 | 13.01
 14 |                            ->  Hash                            |     21 |            |      12 | 13.13
 15 |                               ->  Seq Scan on public.supplier  |     20 |            |      12 | 13.13
 16 |                      ->  Hash                                  |     21 |            |      34 | 13.13
 17 |                         ->  Seq Scan on public.partsupp        |     20 |            |      34 | 13.13
```

13.5.6 Case: Adjusting the Partial Clustering Key

Partial Cluster Key (PCK) is an index technology that uses min/max indexes to quickly scan base tables in column storage. Partial cluster key can specify multiple columns, but you are advised to specify no more than two columns. It can be used to accelerated queries on large column-store tables.

Before Optimization

Create a column-store table **orders_no_pck** without partial clustering (PCK). The table is defined as follows:

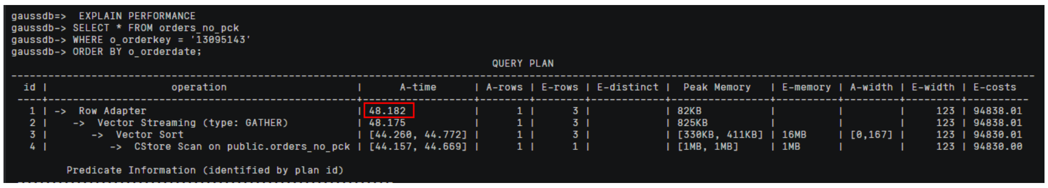


```
                              pg_get_tabledef
----------------------------------------------------------------------------
 SET search_path = dbadmin;                                                +
 CREATE  TABLE orders_no_pck (                                             +
         o_orderkey bigint NOT NULL,                                       +
         o_custkey bigint NOT NULL,                                        +
         o_orderstatus character(1) NOT NULL,                              +
         o_totalprice numeric(15,2) NOT NULL,                              +
         o_orderdate timestamp(0) without time zone NOT NULL,              +
         o_orderpriority character(15) NOT NULL,                           +
         o_clerk character(15) NOT NULL,                                   +
         o_shippriority bigint NOT NULL,                                   +
         o_comment character varying(79) NOT NULL                          +
 )                                                                         +
 WITH (orientation=column, compression=low, colversion=2.0, enable_delta=false)+
 DISTRIBUTE BY HASH(o_orderkey)                                            +
 TO GROUP group_version1;
(1 row)
```

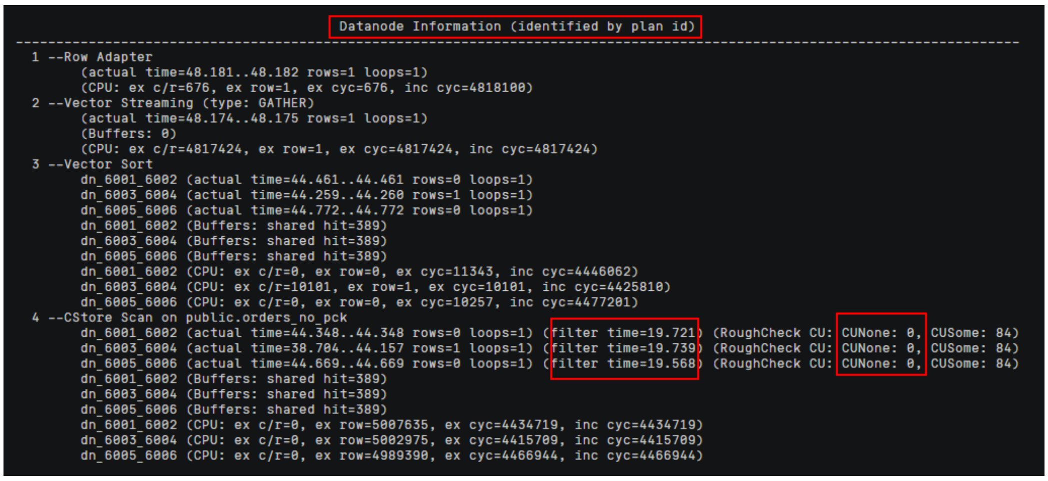
Run the following SQL statement to query the execution plan of a point query:

```
EXPLAIN PERFORMANCE
SELECT * FROM orders_no_pck
WHERE o_orderkey = '13095143'
ORDER BY o_orderdate;
```

As shown in the following figure, the execution time is 48 ms. Check **Datanode Information**. It is found that the filter time is 19 ms and the CUNone ratio is 0.



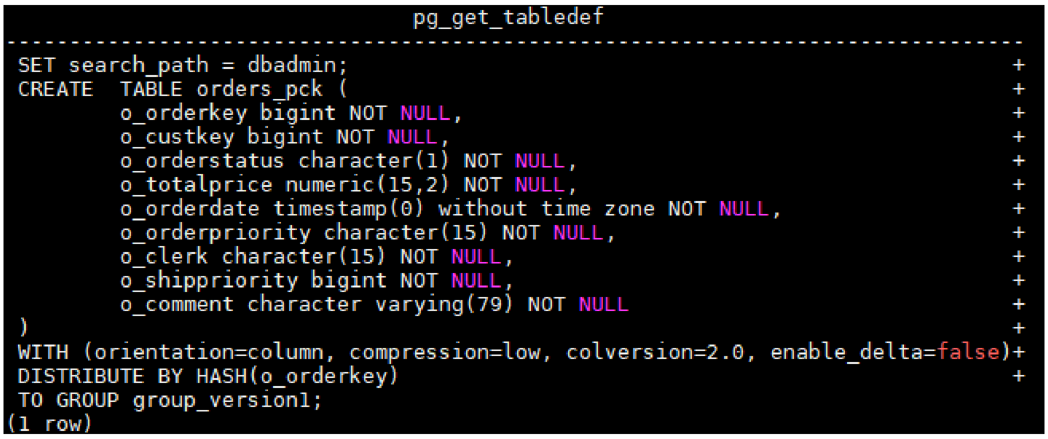
```
gaussdb=>  EXPLAIN PERFORMANCE
gaussdb-> SELECT * FROM orders_no_pck
gaussdb-> WHERE o_orderkey = '13095143'
gaussdb-> ORDER BY o_orderdate;
                                                                   QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------------------------------
 id |                    operation                     |       A-time       | A-rows | E-rows | E-distinct |  Peak Memory   | E-memory | A-width | E-width | E-costs
----+--------------------------------------------------+--------------------+--------+--------+------------+----------------+----------+---------+---------+----------
  1 | ->  Row Adapter                                  | 48.182             |      1 |      3 |            | 82KB           |          |         |     123 | 94838.01
  2 |    ->  Vector Streaming (type: GATHER)           | 48.175             |      1 |      3 |            | 825KB          |          |         |     123 | 94838.01
  3 |       ->  Vector Sort                            | [44.260, 44.772]   |      1 |      3 |            | [330KB, 411KB] | 16MB     | [0,167] |     123 | 94830.01
  4 |          ->  CStore Scan on public.orders_no_pck | [44.157, 44.669]   |      1 |      1 |            | [1MB, 1MB]     | 1MB      |         |     123 | 94830.00

        Predicate Information (identified by plan id)
 ------------------------------------------------------------
```



```
                                    Datanode Information (identified by plan id)
------------------------------------------------------------------------------------------------------------------------------
  1 --Row Adapter
        (actual time=48.181..48.182 rows=1 loops=1)
        (CPU: ex c/r=676, ex row=1, ex cyc=676, inc cyc=4818100)
  2 --Vector Streaming (type: GATHER)
        (actual time=48.174..48.175 rows=1 loops=1)
        (Buffers: 0)
        (CPU: ex c/r=4817424, ex row=1, ex cyc=4817424, inc cyc=4817424)
  3 --Vector Sort
        dn_6001_6002 (actual time=44.461..44.461 rows=0 loops=1)
        dn_6003_6004 (actual time=44.259..44.260 rows=1 loops=1)
        dn_6005_6006 (actual time=44.772..44.772 rows=0 loops=1)
        dn_6001_6002 (Buffers: shared hit=389)
        dn_6003_6004 (Buffers: shared hit=389)
        dn_6005_6006 (Buffers: shared hit=389)
        dn_6001_6002 (CPU: ex c/r=0, ex row=0, ex cyc=11343, inc cyc=4446062)
        dn_6003_6004 (CPU: ex c/r=10101, ex row=1, ex cyc=10101, inc cyc=4425810)
        dn_6005_6006 (CPU: ex c/r=0, ex row=0, ex cyc=10257, inc cyc=4477201)
  4 --CStore Scan on public.orders_no_pck
        dn_6001_6002 (actual time=44.348..44.348 rows=0 loops=1) (filter time=19.721) (RoughCheck CU: CUNone: 0, CUSome: 84)
        dn_6003_6004 (actual time=38.704..44.157 rows=1 loops=1) (filter time=19.739) (RoughCheck CU: CUNone: 0, CUSome: 84)
        dn_6005_6006 (actual time=44.669..44.669 rows=0 loops=1) (filter time=19.568) (RoughCheck CU: CUNone: 0, CUSome: 84)
        dn_6001_6002 (Buffers: shared hit=389)
        dn_6003_6004 (Buffers: shared hit=389)
        dn_6005_6006 (Buffers: shared hit=389)
        dn_6001_6002 (CPU: ex c/r=0, ex row=5007635, ex cyc=4434719, inc cyc=4434719)
        dn_6003_6004 (CPU: ex c/r=0, ex row=5002975, ex cyc=4415709, inc cyc=4415709)
        dn_6005_6006 (CPU: ex c/r=0, ex row=4989390, ex cyc=4466944, inc cyc=4466944)
```

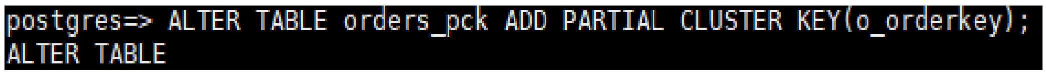
After Optimization

The created column-store table **orders_pck** is defined as follows:



```
                                 pg_get_tabledef
-----------------------------------------------------------------------------------
 SET search_path = dbadmin;                                                       +
 CREATE  TABLE orders_pck (                                                       +
         o_orderkey bigint NOT NULL,                                              +
         o_custkey bigint NOT NULL,                                               +
         o_orderstatus character(1) NOT NULL,                                     +
         o_totalprice numeric(15,2) NOT NULL,                                     +
         o_orderdate timestamp(0) without time zone NOT NULL,                     +
         o_orderpriority character(15) NOT NULL,                                  +
         o_clerk character(15) NOT NULL,                                          +
         o_shippriority bigint NOT NULL,                                          +
         o_comment character varying(79) NOT NULL                                 +
 )                                                                                +
 WITH (orientation=column, compression=low, colversion=2.0, enable_delta=false)+
 DISTRIBUTE BY HASH(o_orderkey)                                                   +
 TO GROUP group_version1;
(1 row)
```

Use **ALTER TABLE** to set the **o_orderkey** field to **PCK**:

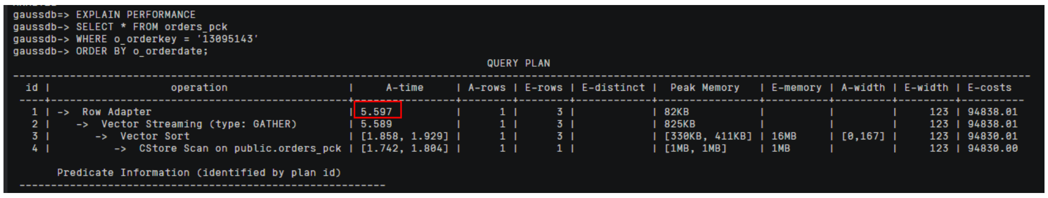


```
postgres=> ALTER TABLE orders_pck ADD PARTIAL CLUSTER KEY(o_orderkey);
ALTER TABLE
```

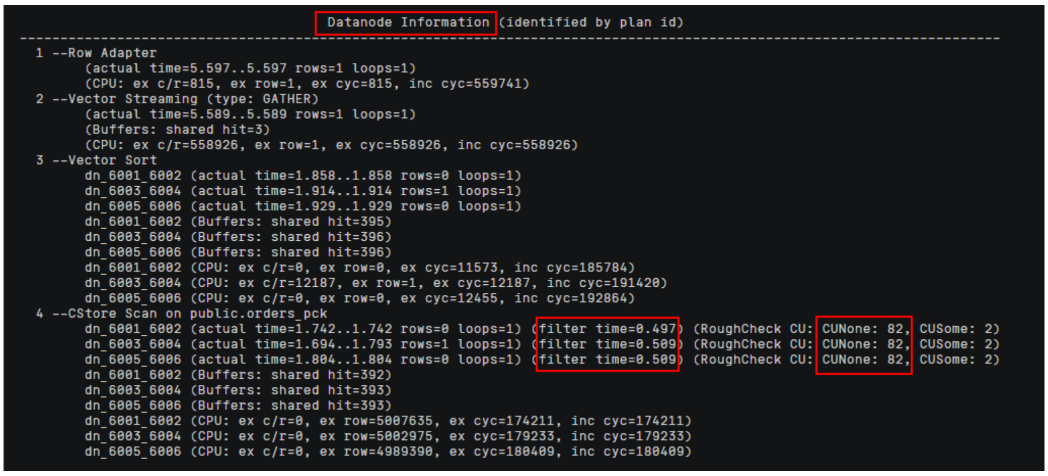
Run the following SQL statement to query the execution plan of the same point query SQL statement again:

```
EXPLAIN PERFORMANCE
SELECT * FROM orders_pck
WHERE o_orderkey = '13095143'
ORDER BY o_orderdate;
```

As shown in the following figure, the execution time is 5 ms. Check **Datanode Information**. It is found that the filter time is 0.5 ms and the CUNone ratio is 82. The higher the CUNone ratio, the higher performance that the PCK will bring.



```
gaussdb=> EXPLAIN PERFORMANCE
gaussdb-> SELECT * FROM orders_pck
gaussdb-> WHERE o_orderkey = '13095143'
gaussdb-> ORDER BY o_orderdate;
                                                            QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------------------
 id |                 operation                  |      A-time      | A-rows | E-rows | E-distinct |   Peak Memory    | E-memory | A-width | E-width | E-costs
----+--------------------------------------------+------------------+--------+--------+------------+------------------+----------+---------+---------+----------
  1 | ->  Row Adapter                            | 5.597            |      1 |      3 |            | 82KB             |          |         |     123 | 94838.01
  2 |    ->  Vector Streaming (type: GATHER)     | 5.589            |      1 |      3 |            | 825KB            |          |         |     123 | 94838.01
  3 |       ->  Vector Sort                      | [1.858, 1.929]   |      1 |      3 |            | [330KB, 411KB]   | 16MB     | [0,167] |     123 | 94830.01
  4 |          ->  CStore Scan on public.orders_pck | [1.742, 1.804] |      1 |      1 |            | [1MB, 1MB]       | 1MB      |         |     123 | 94830.00

       Predicate Information (identified by plan id)
 ----------------------------------------------------------
```



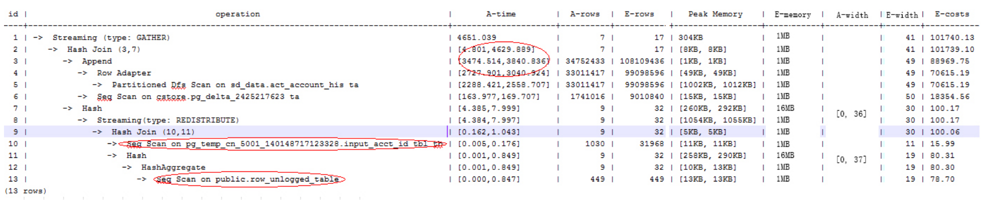
```
                                              Datanode Information (identified by plan id)
 ----------------------------------------------------------------------------------------------------------------------
   1 --Row Adapter
         (actual time=5.597..5.597 rows=1 loops=1)
         (CPU: ex c/r=815, ex row=1, ex cyc=815, inc cyc=559741)
   2 --Vector Streaming (type: GATHER)
         (actual time=5.589..5.589 rows=1 loops=1)
         (Buffers: shared hit=3)
         (CPU: ex c/r=558926, ex row=1, ex cyc=558926, inc cyc=558926)
   3 --Vector Sort
         dn_6001_6002 (actual time=1.858..1.858 rows=0 loops=1)
         dn_6003_6004 (actual time=1.914..1.914 rows=1 loops=1)
         dn_6005_6006 (actual time=1.929..1.929 rows=0 loops=1)
         dn_6001_6002 (Buffers: shared hit=395)
         dn_6003_6004 (Buffers: shared hit=396)
         dn_6005_6006 (Buffers: shared hit=396)
         dn_6001_6002 (CPU: ex c/r=0, ex row=0, ex cyc=11573, inc cyc=185784)
         dn_6003_6004 (CPU: ex c/r=12187, ex row=1, ex cyc=12187, inc cyc=191420)
         dn_6005_6006 (CPU: ex c/r=0, ex row=0, ex cyc=12455, inc cyc=192864)
   4 --CStore Scan on public.orders_pck
         dn_6001_6002 (actual time=1.742..1.742 rows=0 loops=1) (filter time=0.497) (RoughCheck CU: CUNone: 82, CUSome: 2)
         dn_6003_6004 (actual time=1.694..1.793 rows=1 loops=1) (filter time=0.509) (RoughCheck CU: CUNone: 82, CUSome: 2)
         dn_6005_6006 (actual time=1.804..1.804 rows=0 loops=1) (filter time=0.509) (RoughCheck CU: CUNone: 82, CUSome: 2)
         dn_6001_6002 (Buffers: shared hit=392)
         dn_6003_6004 (Buffers: shared hit=393)
         dn_6005_6006 (Buffers: shared hit=393)
         dn_6001_6002 (CPU: ex c/r=0, ex row=5007635, ex cyc=174211, inc cyc=174211)
         dn_6003_6004 (CPU: ex c/r=0, ex row=5002975, ex cyc=179233, inc cyc=179233)
         dn_6005_6006 (CPU: ex c/r=0, ex row=4989390, ex cyc=180409, inc cyc=180409)
```

13.5.7 Case: Adjusting the Table Storage Mode in a Medium Table

In GaussDB(DWS), row-store tables use the row execution engine, and column-store tables use the column execution engine. If both row-store table and column-store tables exist in a SQL statement, the system will automatically select the row execution engine. The performance of a column execution engine (except for the indexscan related operators) is much better than that of a row execution engine. Therefore, a column-store table is recommended. This is important for some medium result set dumping tables, and you need to select a proper table storage type.

Before Optimization

During the test at a site, if the following execution plan is performed, the customer expects that the performance can be improved and the result can be returned within 3s.

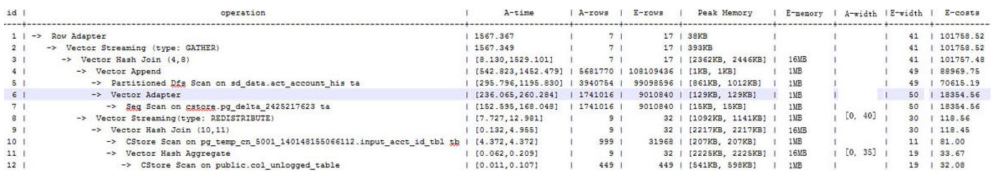


```
 id |                              operation                                      |        A-time         |  A-rows  |  E-rows   |   Peak Memory      | E-memory |  A-width | E-width |  E-costs
----+-----------------------------------------------------------------------------+-----------------------+----------+-----------+--------------------+----------+----------+---------+-----------
  1 | ->  Streaming (type: GATHER)                                                | 4651.039              |        7 |        17 | 304KB              | 1MB      |          |      41 | 101740.13
  2 |    ->  Hash Join (3,7)                                                      | [4.801,4629.889]      |        7 |        17 | [8KB, 8KB]         | 1MB      |          |      41 | 101739.10
  3 |       ->  Append                                                            | [3474.514,3840.836]   | 34752433 | 108109436 | [1KB, 1KB]         | 1MB      |          |      49 | 88969.75
  4 |          ->  Row Adapter                                                    | [2727.901,3040.924]   | 33011417 |  99098596 | [49KB, 49KB]       | 1MB      |          |      49 | 70615.19
  5 |             ->  Partitioned Dfs Scan on sd_data.act_account_his ta          | [2288.421,2558.707]   | 33011417 |  99098596 | [1002KB, 1012KB]   | 1MB      |          |      49 | 70615.19
  6 |          ->  Seq Scan on cstore.pg_delta_2425217623 ta                      | [163.977,169.707]     |  1741016 |   9010840 | [15KB, 15KB]       | 1MB      |          |      50 | 18354.56
  7 |       ->  Hash                                                              | [4.385,7.999]         |        9 |        32 | [260KB, 292KB]     | 16MB     |          |      30 | 100.17
  8 |          ->  Streaming(type: REDISTRIBUTE)                                  | [4.384,7.997]         |        9 |        32 | [1054KB, 1055KB]   | 1MB      | [0, 36]  |      30 | 100.17
  9 |             ->  Hash Join (10,11)                                           | [0.162,1.043]         |        9 |        32 | [5KB, 5KB]         | 1MB      |          |      30 | 100.06
 10 |                ->  Seq Scan on pg_temp_cn_5001_140148717123328.input_acct_id_tbl tb | [0.005,0.176] |     1030 |     31968 | [11KB, 11KB]       | 1MB      |          |      11 | 15.99
 11 |                ->  Hash                                                     | [0.001,0.849]         |        9 |        32 | [258KB, 290KB]     | 16MB     |          |      19 | 80.31
 12 |                   ->  HashAggregate                                         | [0.001,0.849]         |        9 |        32 | [10KB, 13KB]       | 1MB      | [0, 37]  |      19 | 80.30
 13 |                      ->  Seq Scan on public.row_unlogged_table              | [0.000,0.847]         |      449 |       449 | [13KB, 13KB]       | 1MB      |          |      19 | 78.70
(13 rows)
```

After Optimization

It is found that the row engine is used after analysis, because both the temporary plan table input_acct_id_tbl and the medium result dumping table row_unlogged_table use a row-store table.

After the two tables are changed into column-store tables, the system performance is improved and the result is returned by 1.6s.



```
 id |                              operation                                      |        A-time         |  A-rows  |  E-rows   |   Peak Memory      | E-memory |  A-width | E-width |  E-costs
----+-----------------------------------------------------------------------------+-----------------------+----------+-----------+--------------------+----------+----------+---------+-----------
  1 | ->  Row Adapter                                                             | 1567.367              |        7 |        17 | 38KB               |          |          |      41 | 101758.52
  2 |    ->  Vector Streaming (type: GATHER)                                      | 1567.349              |        7 |        17 | 393KB              |          |          |      41 | 101758.52
  3 |       ->  Vector Hash Join (4,8)                                            | [8.130,1529.101]      |        7 |        17 | [2362KB, 2446KB]   | 16MB     |          |      41 | 101757.48
  4 |          ->  Vector Append                                                  | [542.823,1452.479]    |  5681770 | 108109436 | [1KB, 1KB]         | 1MB      |          |      49 | 88969.75
  5 |             ->  Partitioned Dfs Scan on sd_data.act_account_his ta          | [295.796,1195.830]    |  3940754 |  99098596 | [861KB, 1012KB]    | 1MB      |          |      49 | 70615.19
  6 |             ->  Vector Adapter                                              | [236.065,260.284]     |  1741016 |   9010840 | [129KB, 129KB]     | 1MB      |          |      50 | 18354.56
  7 |                ->  Seq Scan on cstore.pg_delta_2425217623 ta                | [152.595,168.048]     |  1741016 |   9010840 | [15KB, 15KB]       | 1MB      |          |      50 | 18354.56
  8 |          ->  Vector Streaming(type: REDISTRIBUTE)                           | [7.727,12.981]        |        9 |        32 | [1092KB, 1141KB]   | 1MB      | [0, 40]  |      30 | 118.56
  9 |             ->  Vector Hash Join (10,11)                                    | [0.132,4.955]         |        9 |        32 | [2217KB, 2217KB]   | 16MB     |          |      30 | 118.45
 10 |                ->  CStore Scan on pg_temp_cn_5001_140148155066112.input_acct_id_tbl tb | [4.372,4.372] |  999 |     31968 | [207KB, 207KB]     | 1MB      |          |      11 | 81.00
 11 |                ->  Vector Hash Aggregate                                    | [0.062,0.209]         |        9 |        32 | [2225KB, 2225KB]   | 16MB     | [0, 35]  |      19 | 33.67
 12 |                   ->  CStore Scan on public.col_unlogged_table              | [0.011,0.107]         |      449 |       449 | [541KB, 598KB]     | 1MB      |          |      19 | 32.08
```

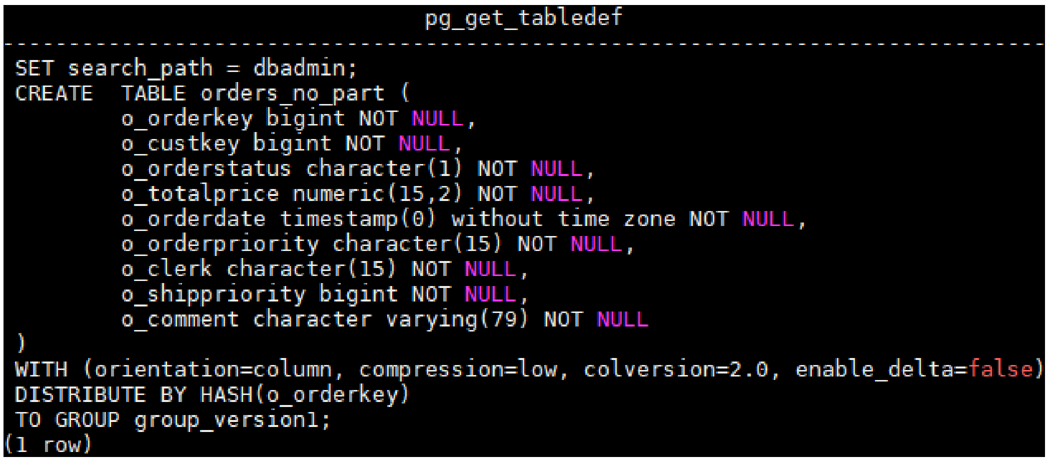
13.5.8 Case: Reconstructing Partition Tables

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. Generally, partitioning is applied to tables that have obvious ranges. Partitions on such tables allow scanning on a small part of data, improving the query performance.

During query, partition pruning is used to minimize bottom-layer data scanning to narrow down the overall scope of scanning in a table. Partition pruning means that the optimizer can automatically extract partitions to be scanned based on the partition key specified in the **FROM** and **WHERE** statements. This avoids full table scanning, reduces the number of data blocks to be scanned, and improves performance.

Before Optimization

Create a non-partition table **orders_no_part**. The table definition is as follows:

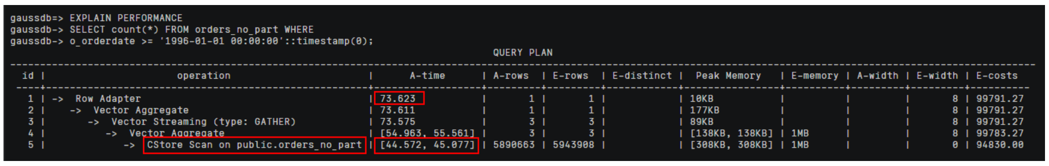


```
                          pg_get_tabledef
--------------------------------------------------------------------
 SET search_path = dbadmin;
 CREATE  TABLE orders_no_part (
         o_orderkey bigint NOT NULL,
         o_custkey bigint NOT NULL,
         o_orderstatus character(1) NOT NULL,
         o_totalprice numeric(15,2) NOT NULL,
         o_orderdate timestamp(0) without time zone NOT NULL,
         o_orderpriority character(15) NOT NULL,
         o_clerk character(15) NOT NULL,
         o_shippriority bigint NOT NULL,
         o_comment character varying(79) NOT NULL
 )
 WITH (orientation=column, compression=low, colversion=2.0, enable_delta=false)
 DISTRIBUTE BY HASH(o_orderkey)
 TO GROUP group_version1;
(1 row)
```

Run the following SQL statement to query the execution plan of the non-partition table:

```
EXPLAIN PERFORMANCE
SELECT count(*) FROM orders_no_part WHERE
o_orderdate >= '1996-01-01 00:00:00'::timestamp(0);
```

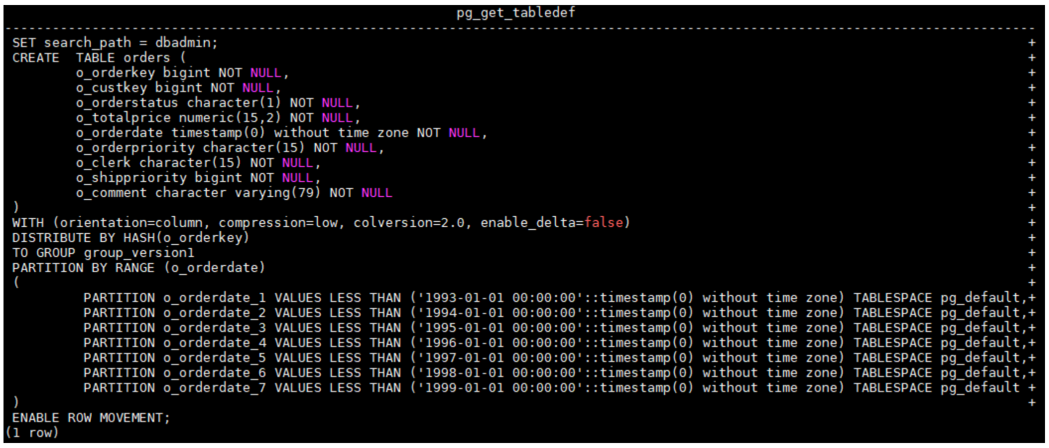
As shown in the following figure, the execution time is 73 milliseconds, and the full table scanning time is 44 to 45 milliseconds.



```
gaussdb=> EXPLAIN PERFORMANCE
gaussdb-> SELECT count(*) FROM orders_no_part WHERE
gaussdb-> o_orderdate >= '1996-01-01 00:00:00'::timestamp(0);
                                                                     QUERY PLAN
----------------------------------------------------------------------------------------------------------------------------------------------------------------
 id |                      operation                       |       A-time       | A-rows  | E-rows  | E-distinct |  Peak Memory    | E-memory | A-width | E-width | E-costs
----+------------------------------------------------------+--------------------+---------+---------+------------+-----------------+----------+---------+---------+----------
  1 | ->  Row Adapter                                      | 73.623             |       1 |       1 |            | 10KB            |          |         |       8 | 99791.27
  2 |    ->  Vector Aggregate                              | 73.611             |       1 |       1 |            | 177KB           |          |         |       8 | 99791.27
  3 |       ->  Vector Streaming (type: GATHER)            | 73.575             |       3 |       3 |            | 89KB            |          |         |       8 | 99791.27
  4 |          ->  Vector Aggregate                        | [54.963, 55.561]   |       3 |       3 |            | [138KB, 138KB]  | 1MB      |         |       8 | 99783.27
  5 |             ->  CStore Scan on public.orders_no_part | [44.572, 45.077]   | 5890663 | 5943908 |            | [308KB, 308KB]  | 1MB      |         |       0 | 94830.00
```

After Optimization

Create a partitioned table **orders**. The table is defined as follows:

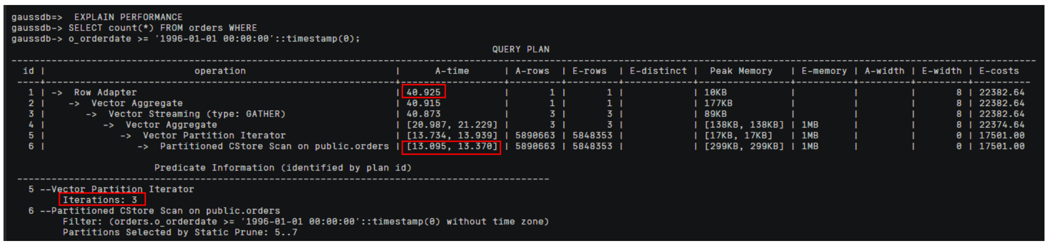


```
                                           pg_get_tabledef
-------------------------------------------------------------------------------------------------------------
 SET search_path = dbadmin;                                                                                 +
 CREATE  TABLE orders (                                                                                     +
         o_orderkey bigint NOT NULL,                                                                        +
         o_custkey bigint NOT NULL,                                                                         +
         o_orderstatus character(1) NOT NULL,                                                               +
         o_totalprice numeric(15,2) NOT NULL,                                                               +
         o_orderdate timestamp(0) without time zone NOT NULL,                                               +
         o_orderpriority character(15) NOT NULL,                                                            +
         o_clerk character(15) NOT NULL,                                                                    +
         o_shippriority bigint NOT NULL,                                                                    +
         o_comment character varying(79) NOT NULL                                                           +
 )                                                                                                          +
 WITH (orientation=column, compression=low, colversion=2.0, enable_delta=false)                             +
 DISTRIBUTE BY HASH(o_orderkey)                                                                             +
 TO GROUP group_version1                                                                                    +
 PARTITION BY RANGE (o_orderdate)                                                                           +
 (                                                                                                          +
          PARTITION o_orderdate_1 VALUES LESS THAN ('1993-01-01 00:00:00'::timestamp(0) without time zone) TABLESPACE pg_default,+
          PARTITION o_orderdate_2 VALUES LESS THAN ('1994-01-01 00:00:00'::timestamp(0) without time zone) TABLESPACE pg_default,+
          PARTITION o_orderdate_3 VALUES LESS THAN ('1995-01-01 00:00:00'::timestamp(0) without time zone) TABLESPACE pg_default,+
          PARTITION o_orderdate_4 VALUES LESS THAN ('1996-01-01 00:00:00'::timestamp(0) without time zone) TABLESPACE pg_default,+
          PARTITION o_orderdate_5 VALUES LESS THAN ('1997-01-01 00:00:00'::timestamp(0) without time zone) TABLESPACE pg_default,+
          PARTITION o_orderdate_6 VALUES LESS THAN ('1998-01-01 00:00:00'::timestamp(0) without time zone) TABLESPACE pg_default,+
          PARTITION o_orderdate_7 VALUES LESS THAN ('1999-01-01 00:00:00'::timestamp(0) without time zone) TABLESPACE pg_default +
 )                                                                                                          +
 ENABLE ROW MOVEMENT;
(1 row)
```

Run the SQL statement again to query the execution plan of the partitioned table. The execution time is 40 ms, in which the table scanning time is only 13 ms. The smaller the value of **Iterations**, the better the partition pruning effect.

```
EXPLAIN PERFORMANCE
SELECT count(*) FROM orders_no_part WHERE
o_orderdate >= '1996-01-01 00:00:00'::timestamp(0);
```

As shown in the following figure, the execution time is 40 milliseconds, and the table scanning time is only 13 milliseconds. A smaller **Iterations** value indicates a better partition pruning effect.



```
gaussdb=>  EXPLAIN PERFORMANCE
gaussdb-> SELECT count(*) FROM orders WHERE
gaussdb-> o_orderdate >= '1996-01-01 00:00:00'::timestamp(0);
                                                                                   QUERY PLAN
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
 id |                          operation                       |       A-time       | A-rows  | E-rows  | E-distinct |  Peak Memory    | E-memory | A-width | E-width | E-costs
----+-----------------------------------------------------------+--------------------+---------+---------+------------+-----------------+----------+---------+---------+---------
  1 | ->  Row Adapter                                           | 40.925             |       1 |       1 |            | 10KB            |          |         |       8 | 22382.64
  2 |    ->  Vector Aggregate                                   | 40.915             |       1 |       1 |            | 177KB           |          |         |       8 | 22382.64
  3 |       ->  Vector Streaming (type: GATHER)                 | 40.873             |       3 |       3 |            | 89KB            |          |         |       8 | 22382.64
  4 |          ->  Vector Aggregate                             | [20.987, 21.229]   |       3 |       3 |            | [138KB, 138KB]  | 1MB      |         |       8 | 22374.64
  5 |             ->  Vector Partition Iterator                 | [13.734, 13.939]   | 5890663 | 5848353 |            | [17KB, 17KB]    | 1MB      |         |       0 | 17501.00
  6 |                ->  Partitioned CStore Scan on public.orders | [13.095, 13.370] | 5890663 | 5848353 |            | [299KB, 299KB]  | 1MB      |         |       0 | 17501.00

                            Predicate Information (identified by plan id)
 ---------------------------------------------------------------------------------------------------------
   5 --Vector Partition Iterator
         Iterations: 3
   6 --Partitioned CStore Scan on public.orders
         Filter: (orders.o_orderdate >= '1996-01-01 00:00:00'::timestamp(0) without time zone)
         Partitions Selected by Static Prune: 5..7
```

13.5.9 Case: Adjusting the GUC Parameter best_agg_plan

Symptom

The t1 table is defined as follows:

```
create table t1(a int, b int, c int) distribute by hash(a);
```

Assume that the distribution column of the result set provided by the agg lower-layer operator is setA, and the group by column of the agg operation is setB, the agg operations can be performed in two scenarios in the stream framework.

Scenario 1: setA is a subset of setB.

In this scenario, the aggregation result of the lower-layer result set is the correct result, which can be directly used by the upper-layer operator. For details, see the following figure:

```
explain select a, count(1) from t1 group by a;
 id |          operation           | E-rows | E-width | E-costs
----+------------------------------+--------+---------+---------
  1 | ->  Streaming (type: GATHER) |     30 |       4 | 15.56
  2 |    ->  HashAggregate         |     30 |       4 | 14.31
  3 |       ->  Seq Scan on t1     |     30 |       4 | 14.14
(3 rows)
```

Scenario 2: setA is not a subset of setB.

In this scenario, the Stream execution framework is classified into the following three plans:

hashagg+gather(redistribute)+hashagg

redistribute+hashagg(+gather)

hashagg+redistribute+hashagg(+gather)

GaussDB(DWS) provides the guc parameter **best_agg_plan** to intervene the execution plan, and forces the plan to generate the corresponding execution plan. This parameter can be set to **0**, **1**, **2**, and **3**.

- When the value is set to **1**, the first plan is forcibly generated.
- When the value is set to **2** and if the **group by** column can be redistributed, the second plan is forcibly generated. Otherwise, the first plan is generated.

- When the value is set to **3** and if the **group by** column can be redistributed, the third plan is generated. Otherwise, the first plan is generated.
- When the value is set to **0**, the query optimizer chooses the most optimal plan by the three preceding plans' evaluation cost.

Possible impacts are as follows:

```
set best_agg_plan to 1;
SET
explain select b,count(1) from t1 group by b;
 id |            operation            | E-rows | E-width | E-costs
----+--------------------------------+--------+---------+---------
  1 | ->  HashAggregate              |      8 |       4 | 15.83
  2 |    ->  Streaming (type: GATHER) |     25 |       4 | 15.83
  3 |       ->  HashAggregate        |     25 |       4 | 14.33
  4 |          ->  Seq Scan on t1     |     30 |       4 | 14.14
(4 rows)
set best_agg_plan to 2;
SET
explain select b,count(1) from t1 group by b;
 id |                operation                | E-rows | E-width | E-costs
----+----------------------------------------+--------+---------+---------
  1 | ->  Streaming (type: GATHER)           |     30 |       4 | 15.85
  2 |    ->  HashAggregate                   |     30 |       4 | 14.60
  3 |       ->  Streaming(type: REDISTRIBUTE) |     30 |       4 | 14.45
  4 |          ->  Seq Scan on t1            |     30 |       4 | 14.14
(4 rows)
set best_agg_plan to 3;
SET
explain select b,count(1) from t1 group by b;
 id |                operation                | E-rows | E-width | E-costs
----+----------------------------------------+--------+---------+---------
  1 | ->  Streaming (type: GATHER)           |     30 |       4 | 15.84
  2 |    ->  HashAggregate                   |     30 |       4 | 14.59
  3 |       ->  Streaming(type: REDISTRIBUTE) |     25 |       4 | 14.59
  4 |          ->  HashAggregate            |     25 |       4 | 14.33
  5 |             ->  Seq Scan on t1         |     30 |       4 | 14.14
(5 rows)
```

Summary

Generally, the optimizer chooses an optimal execution plan, but the cost estimation, especially that of the intermediate result set, has large deviations, which may result in large deviations in agg calculation. In this case, you need to use best_agg_plan to adjust the agg calculation model.

When the aggregation convergence ratio is very small, that is, the number of result sets does not become small obviously after the agg operation (5 times is a critical point), you can select the redistribute+hashagg or hashagg+redistribute +hashagg execution mode.

13.5.10 Case: Rewriting SQL Statements and Eliminating Prune Interference

A filter criterion that contains the expression of partition key cannot be used for pruning. As a result, the query statement scans almost all data in the partitioned table.

Before Optimization

t_ddw_f10_op_cust_asset_mon indicates the partitioned table. **year_mth** indicates the partition key. This field is an integer consisting of the **year** and **mth** values.

The following figure shows the tested SQL statements.

```
SELECT
    count(1)
FROM t_ddw_f10_op_cust_asset_mon b1
WHERE b1.year_mth  < substr('20200722',1 ,6 )
AND b1.year_mth + 1 >= substr('20200722',1 ,6 );
```

The test result shows that the table scan of the SQL statement takes 10 seconds. The execution plan of the SQL statement is as follows.

```
EXPLAIN (ANALYZE ON, VERBOSE ON)
SELECT
    count(1)
FROM t_ddw_f10_op_cust_asset_mon b1
WHERE b1.year_mth < substr('20200722',1 ,6 )
AND b1.year_mth + 1 >= cast(substr('20200722',1 ,6 ) AS int);
                                                                        QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
  id |                                operation                                  |       A-time        |  A-rows  |  E-rows  | E-distinct | Peak Memory  | E-memory | A-width | E-width |  E-costs
 ----+--------------------------------------------------------------------------+-----------------------+----------+----------+------------+--------------+----------+---------+---------+-----------
   1 | ->  Aggregate                                                             | 10662.260            |        1 |        1 |            | 32KB         |          |         |       8 | 593656.42
   2 |    ->  Streaming (type: GATHER)                                           | 10662.172            |        4 |        4 |            | 136KB        |          |         |       8 | 593656.42
   3 |       ->  Aggregate                                                       | [9692.785, 10656.068] |        4 |        4 |            | [24KB, 24KB] | 1MB      |         |       8 | 593646.42
   4 |          ->  Partition Iterator                                           | [8787.198, 9629.138]  | 16384000 | 32752850 |            | [16KB, 16KB] | 1MB      |         |       0 | 573175.88
   5 |             ->  Partitioned Seq Scan on public.t_ddw_f10_op_cust_asset_mon b1 | [8365.655, 9152.115]  | 16384000 | 32752850 |            | [32KB, 32KB] | 1MB      |         |       0 | 573175.88

                              SQL Diagnostic Information
 ------------------------------------------------------------------------------------------
 Partitioned table unprunable Qual
        table public.t_ddw_f10_op_cust_asset_mon b1:
        left side of expression "((year_mth + 1) > 202008)" invokes function-call/type-conversion

                 Predicate Information (identified by plan id)
 ---------------------------------------------------------------------------------
   4 --Partition Iterator
        Iterations: 6
   5 --Partitioned Seq Scan on public.t_ddw_f10_op_cust_asset_mon b1
        Filter: ((b1.year_mth < 202007::bigint) AND ((b1.year_mth + 1) >= 202007))
        Rows Removed by Filter: 81920000
        Partitions Selected by Static Prune: 1..6
```

After Optimization

After analyzing the execution plan of the statement and checking the SQL self-diagnosis information in the execution plan, the following diagnosis information is found:

```
                              SQL Diagnostic Information
 ------------------------------------------------------------------------------------------
 Partitioned table unprunable Qual
        table public.t_ddw_f10_op_cust_asset_mon b1:
        left side of expression "((year_mth + 1) > 202008)" invokes function-call/type-conversion
```

The filter criterion contains the expression **(year_mth + 1) > 202008**. A filter criterion that contains the expression of partition key cannot be used for pruning. As a result, the query statement scans almost all data in the partitioned table.

Compared with the original SQL statement, the expression **(year_mth + 1) > 202008** is derived from the expression **b1.year_mth + 1 > substr('20200822',1 ,6)**. Based on the diagnosis information, the SQL statement is modified as follows.

```
SELECT
   count(1)
FROM t_ddw_f10_op_cust_asset_mon b1
WHERE b1.year_mth <= substr('20200822',1 ,6 )
AND b1.year_mth > cast(substr('20200822',1 ,6 ) AS int) - 1;
```

After the modification, the SQL statement execution information is as follows. The alarm indicating that the pruning is not performed is cleared. After the pruning, the score of the partition to be scanned is 1, and the execution time is shortened from 10 seconds to 3 seconds.

```
EXPLAIN (analyze ON, verbose ON)
SELECT
   count(1)
FROM t_ddw_f10_op_cust_asset_mon b1
WHERE b1.year_mth < substr('20200722',1 ,6 )
AND b1.year_mth >= cast(substr('20200722',1 ,6 ) AS int) - 1;
                                                         QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------
  id |                              operation                              |       A-time        | A-rows  |  E-rows  | E-
distinct | Peak Memory  | E-memory | A-width | E-width |  E-costs
 ----+--------------------------------------------------------------------------------+----------------------+----------
+----------+------------+--------------+----------+---------+---------+-----------
   1 | ->  Aggregate                                                      | 3009.796            |        1 |        1 |            |
32KB         |          |         |       8 | 501541.70
   2 |    ->  Streaming (type: GATHER)                                          | 3009.718            |        4 |        4
|           | 136KB        |          |         |       8 | 501541.70
   3 |       ->  Aggregate                                               | [2675.509, 3003.298] |        4 |        4
|           | [24KB, 24KB] | 1MB      |         |       8 | 501531.70
   4 |          ->  Partition Iterator                                   | [1820.725, 2053.836] | 16384000 |
16380697 |           | [16KB, 16KB] | 1MB      |         |       0 | 491293.75
   5 |             ->  Partitioned Seq Scan on public.t_ddw_f10_op_cust_asset_mon b1 | [1420.972, 1590.083] |
16384000 | 16380697 |           | [16KB, 16KB] | 1MB      |         |       0 | 491293.75

              Predicate Information (identified by plan id)
 ---------------------------------------------------------------------------
   4 --Partition Iterator
         Iterations: 1
   5 --Partitioned Seq Scan on public.t_ddw_f10_op_cust_asset_mon b1
         Filter: ((b1.year_mth < 202007::bigint) AND (b1.year_mth >= 202006))
         Partitions Selected by Static Prune: 6
```

13.5.11 Case: Rewriting SQL Statements and Deleting in-clause

Before Optimization

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

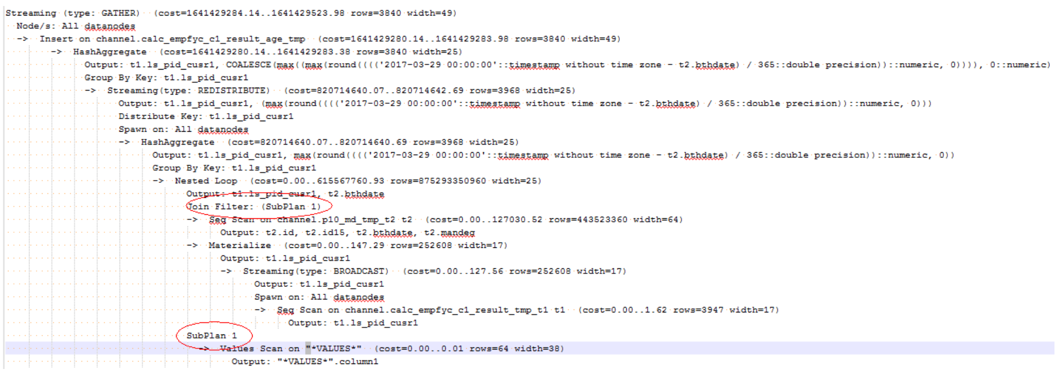
```
select
count(1)
from calc_empfyc_c1_result_tmp_t1
where ls_pid_cusr1 in any('20120405', '20130405');
```

Some special usages are as follows:

```
SELECT
ls_pid_cusr1,COALESCE(max(round((current_date-bthdate)/365)),0)
FROM calc_empfyc_c1_result_tmp_t1 t1,p10_md_tmp_t2 t2
WHERE t1.ls_pid_cusr1 = any(values(id),(id15))
GROUP BY ls_pid_cusr1;
```

Where **id** and **id15** are columns of p10_md_tmp_t2. ls_pid_cusr1 = any(values(id),(id15)) equals t1. ls_pid_cusr1 = id or t1. ls_pid_cusr1 = id15.

Therefore, join-condition is essentially an inequality, and nestloop must be used for this join operation. The execution plan is as follows:



```
Streaming (type: GATHER)  (cost=1641429284.14..1641429523.98 rows=3840 width=49)
  Node/s: All datanodes
  ->  Insert on channel.calc_empfyc_c1_result_age_tmp  (cost=1641429280.14..1641429283.98 rows=3840 width=49)
        ->  HashAggregate  (cost=1641429280.14..1641429283.38 rows=3840 width=25)
              Output: t1.ls_pid_cusr1, COALESCE(max((max(round(((('2017-03-29 00:00:00'::timestamp without time zone - t2.bthdate) / 365::double precision))::numeric, 0)))), 0::numeric)
              Group By Key: t1.ls_pid_cusr1
              ->  Streaming(type: REDISTRIBUTE)  (cost=820714640.07..820714642.69 rows=3968 width=25)
                    Output: t1.ls_pid_cusr1, (max(round(((('2017-03-29 00:00:00'::timestamp without time zone - t2.bthdate) / 365::double precision))::numeric, 0)))
                    Distribute Key: t1.ls_pid_cusr1
                    Spawn on: All datanodes
                    ->  HashAggregate  (cost=820714640.07..820714640.69 rows=3968 width=25)
                          Output: t1.ls_pid_cusr1, max(round(((('2017-03-29 00:00:00'::timestamp without time zone - t2.bthdate) / 365::double precision))::numeric, 0))
                          Group By Key: t1.ls_pid_cusr1
                          ->  Nested Loop  (cost=0.00..615567760.93 rows=875293350960 width=25)
                                Output: t1.ls_pid_cusr1, t2.bthdate
                                Join Filter: (SubPlan 1)
                                ->  Seq Scan on channel.p10_md_tmp_t2 t2  (cost=0.00..127030.52 rows=443523360 width=64)
                                      Output: t2.id, t2.id15, t2.bthdate, t2.mandeg
                                ->  Materialize  (cost=0.00..147.29 rows=252608 width=17)
                                      Output: t1.ls_pid_cusr1
                                      ->  Streaming(type: BROADCAST)  (cost=0.00..127.56 rows=252608 width=17)
                                            Output: t1.ls_pid_cusr1
                                            Spawn on: All datanodes
                                            ->  Seq Scan on channel.calc_empfyc_c1_result_tmp_t1 t1  (cost=0.00..1.62 rows=3947 width=17)
                                                  Output: t1.ls_pid_cusr1
                                SubPlan 1
                                  ->  Values Scan on "*VALUES*"  (cost=0.00..0.01 rows=64 width=38)
                                        Output: "*VALUES*".column1
```

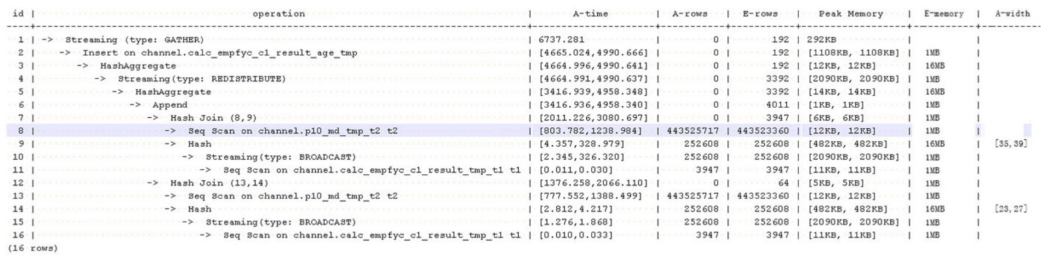
After Optimization

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 hashjoin. From the perspective of semantic equivalence, the SQL statements can be written as follows:

```
select
ls_pid_cusr1,COALESCE(max(round(ym/365)),0)
from
(
        (
                SELECT
                        ls_pid_cusr1,(current_date-bthdate) as ym
                FROM calc_empfyc_c1_result_tmp_t1 t1,p10_md_tmp_t2 t2
                WHERE t1.ls_pid_cusr1 = t2.id and t1.ls_pid_cusr1 != t2.id15
        )
        union all
        (
                SELECT
                        ls_pid_cusr1,(current_date-bthdate) as ym
                FROM calc_empfyc_c1_result_tmp_t1 t1,p10_md_tmp_t2 t2
                WHERE t1.ls_pid_cusr1 = id15
        )
)
GROUP BY ls_pid_cusr1;
```

Note: Use **UNION ALL** instead of **UNION** if possible. **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.

The optimized SQL queries consist of two equivalent join subqueries, and each subquery can be used for hashjoin in this scenario. The optimized execution plan is as follows:



```
 id |                          operation                                        |       A-time        |  A-rows   |  E-rows   |   Peak Memory    | E-memory | A-width
----+---------------------------------------------------------------------------+---------------------+-----------+-----------+------------------+----------+---------
  1 | ->  Streaming (type: GATHER)                                              | 6737.281            |         0 |       192 | 292KB            |          |
  2 |    ->  Insert on channel.calc_empfyc_c1_result_age_tmp                    | [4665.024,4990.666] |         0 |       192 | [1108KB, 1108KB] | 1MB      |
  3 |       ->  HashAggregate                                                   | [4664.996,4990.641] |         0 |       192 | [12KB, 12KB]     | 16MB     |
  4 |          ->  Streaming(type: REDISTRIBUTE)                                | [4664.991,4990.637] |         0 |      3392 | [2090KB, 2090KB] | 1MB      |
  5 |             ->  HashAggregate                                             | [3416.939,4958.348] |         0 |      3392 | [14KB, 14KB]     | 16MB     |
  6 |                ->  Append                                                 | [3416.936,4958.340] |         0 |      4011 | [1KB, 1KB]       | 1MB      |
  7 |                   ->  Hash Join (8,9)                                     | [2011.226,3080.697] |         0 |      3947 | [6KB, 6KB]       | 1MB      |
  8 |                      ->  Seq Scan on channel.p10_md_tmp_t2 t2             | [803.782,1238.984]  | 443525717 | 443523360 | [12KB, 12KB]     | 1MB      |
  9 |                      ->  Hash                                             | [4.357,328.979]     |    252608 |    252608 | [482KB, 482KB]   | 16MB     | [35,39]
 10 |                         ->  Streaming(type: BROADCAST)                    | [2.345,326.320]     |    252608 |    252608 | [2090KB, 2090KB] | 1MB      |
 11 |                            ->  Seq Scan on channel.calc_empfyc_c1_result_tmp_t1 t1 | [0.011,0.030] |  3947 |      3947 | [11KB, 11KB]     | 1MB      |
 12 |                   ->  Hash Join (13,14)                                   | [1376.258,2066.110] |         0 |        64 | [5KB, 5KB]       | 1MB      |
 13 |                      ->  Seq Scan on channel.p10_md_tmp_t2 t2             | [777.552,1388.499]  | 443525717 | 443523360 | [12KB, 12KB]     | 1MB      |
 14 |                      ->  Hash                                             | [2.812,4.217]       |    252608 |    252608 | [482KB, 482KB]   | 16MB     | [23,27]
 15 |                         ->  Streaming(type: BROADCAST)                    | [1.276,1.868]       |    252608 |    252608 | [2090KB, 2090KB] | 1MB      |
 16 |                            ->  Seq Scan on channel.calc_empfyc_c1_result_tmp_t1 t1 | [0.010,0.033] |  3947 |      3947 | [11KB, 11KB]     | 1MB      |
(16 rows)
```

Before the optimization, no result is returned for more than 1 hour. After the optimization, the result is returned within 7s.

13.5.12 Case: Setting Partial Cluster Keys

You can add **PARTIAL CLUSTER KEY**(*column_name*[,...]) to the definition of a column-store table to set one or more columns of this table as partial cluster keys. In this way, each 70 CUs (4.2 million rows) will be sorted based on the cluster keys by default during data import and the value range is narrowed down for each of the new 70 CUs. If the **where** condition in the query statement contains these columns, the filtering performance will be improved.

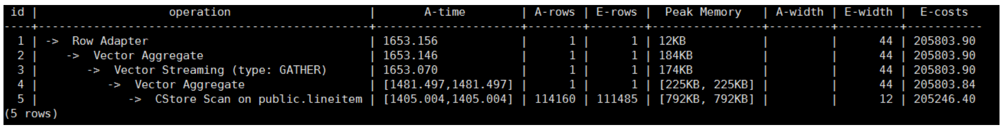
Before Optimization

The partial cluster key is not used. The table is defined 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)
distribute by hash(L_ORDERKEY);

select
sum(l_extendedprice * l_discount) as revenue
from
lineitem
where
l_shipdate >= '1994-01-01'::date
and l_shipdate < '1994-01-01'::date + interval '1 year'
and l_discount between 0.06 - 0.01 and 0.06 + 0.01
and l_quantity < 24;
```

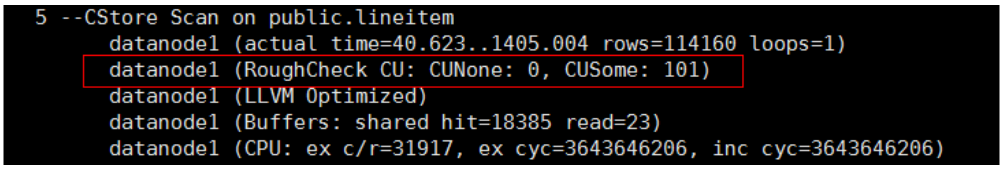
After the data is imported, perform the query and check the execution time.

Figure 13-23 Partial cluster keys not used



```
 id |                  operation                  |        A-time         | A-rows | E-rows |  Peak Memory    | A-width | E-width |  E-costs
----+---------------------------------------------+-----------------------+--------+--------+-----------------+---------+---------+-----------
  1 | ->  Row Adapter                             | 1653.156              |      1 |      1 | 12KB            |         |      44 | 205803.90
  2 |    ->  Vector Aggregate                     | 1653.146              |      1 |      1 | 184KB           |         |      44 | 205803.90
  3 |       ->  Vector Streaming (type: GATHER)   | 1653.070              |      1 |      1 | 174KB           |         |      44 | 205803.90
  4 |          ->  Vector Aggregate               | [1481.497,1481.497]   |      1 |      1 | [225KB, 225KB]  |         |      44 | 205803.84
  5 |             ->  CStore Scan on public.lineitem | [1405.004,1405.004] | 114160 | 111485 | [792KB, 792KB]  |         |      12 | 205246.40
(5 rows)
```

Figure 13-24 CU loading without partial cluster keys



```
  5 --CStore Scan on public.lineitem
        datanode1 (actual time=40.623..1405.004 rows=114160 loops=1)
        datanode1 (RoughCheck CU: CUNone: 0, CUSome: 101)
        datanode1 (LLVM Optimized)
        datanode1 (Buffers: shared hit=18385 read=23)
        datanode1 (CPU: ex c/r=31917, ex cyc=3643646206, inc cyc=3643646206)
```

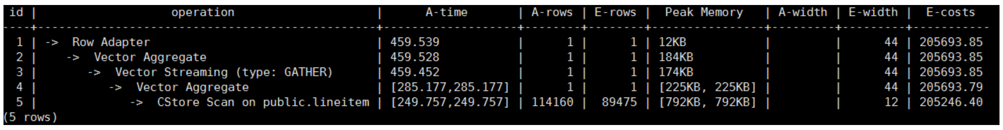
After Optimization

In the **where** condition, both the **l_shipdate** and **l_quantity** columns have a few distinct values, and their values can be used for min/max filtering. Therefore, modify the table definition 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
, partial cluster key(l_shipdate, l_quantity)
)
with (orientation = column)
distribute by hash(L_ORDERKEY);
```

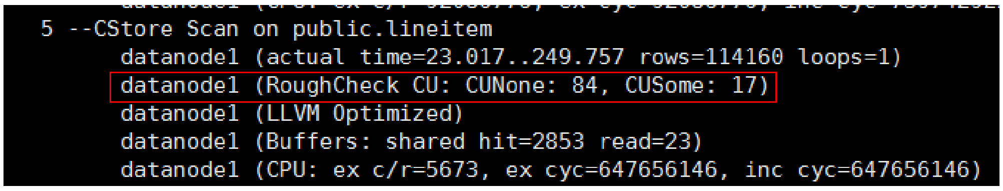
Import the data again, perform the query, and check the execution time.

Figure 13-25 Partial cluster keys used



```
 id |                  operation                  |        A-time         | A-rows | E-rows |  Peak Memory    | A-width | E-width |  E-costs
----+---------------------------------------------+-----------------------+--------+--------+-----------------+---------+---------+-----------
  1 | ->  Row Adapter                             | 459.539               |      1 |      1 | 12KB            |         |      44 | 205693.85
  2 |    ->  Vector Aggregate                     | 459.528               |      1 |      1 | 184KB           |         |      44 | 205693.85
  3 |       ->  Vector Streaming (type: GATHER)   | 459.452               |      1 |      1 | 174KB           |         |      44 | 205693.85
  4 |          ->  Vector Aggregate               | [285.177,285.177]     |      1 |      1 | [225KB, 225KB]  |         |      44 | 205693.79
  5 |             ->  CStore Scan on public.lineitem | [249.757,249.757]   | 114160 |  89475 | [792KB, 792KB]  |         |      12 | 205246.40
(5 rows)
```

Figure 13-26 CU loading with partial cluster keys



```
  5 --CStore Scan on public.lineitem
        datanode1 (actual time=23.017..249.757 rows=114160 loops=1)
        datanode1 (RoughCheck CU: CUNone: 84, CUSome: 17)
        datanode1 (LLVM Optimized)
        datanode1 (Buffers: shared hit=2853 read=23)
        datanode1 (CPU: ex c/r=5673, ex cyc=647656146, inc cyc=647656146)
```

After partial cluster keys are used, the execution time of **5-- CStore Scan on public.lineitem** decreases by 1.2s because 84 CUs are filtered out.

Optimization

- Select partial cluster keys.
  - The following data types support cluster keys: character varying(n), varchar(n), character(n), char(n), text, nvarchar2, timestamp with time zone, timestamp without time zone, date, time without time zone, and time with time zone.
  - Smaller number of distinct values in a partial cluster key generates higher filtering performance.
  - Columns that can filter out larger amount of data is preferentially selected as partial cluster keys.
  - If multiple columns are selected as partial cluster keys, the columns are used in sequence to sort data. You are advised to select a maximum of three columns.
- Modify parameters to reduce the impact of partial cluster keys on the import performance.

  After partial cluster keys are used, data will be sorted when they are imported, affecting the import performance. If all the data can be sorted in the memory, the keys have little impact on import. If some data cannot be sorted in the memory and is written into a temporary file for sorting, the import performance will be greatly affected.

  The memory used for sorting is specified by the **psort_work_mem** parameter. You can set it to a larger value so that the sorting has less impact on the import performance.

  The volume of data to be sorted is specified by the **PARTIAL_CLUSTER_ROWS** parameter of the table. Decreasing the value of this parameter reduces the amount of data to be sorted at a time. **PARTIAL_CLUSTER_ROWS** is usually used along with the **MAX_BATCHROW** parameter. The value of **PARTIAL_CLUSTER_ROWS** must be an integer multiple of the **MAX_BATCHROW** value. **MAX_BATCHROW** specifies the maximum number of rows in a CU.

13.5.13 Case: Converting from NOT IN to NOT EXISTS

nestloop anti join must be used to implement **NOT IN**, while you can use **Hash anti join** to implement **NOT EXISTS**. If no **NULL** value exists in the **JOIN** column, **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 joins** and to improve the query performance.

Before Optimization

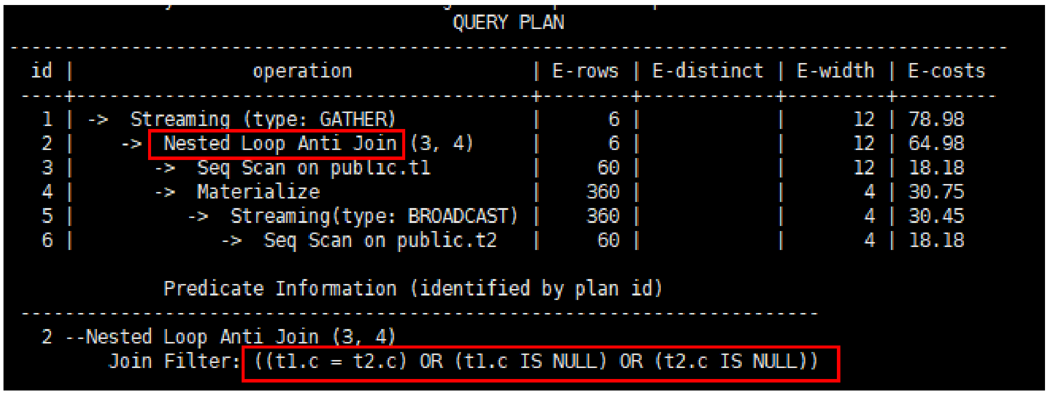
Create two base tables **t1** and **t2**.

```
CREATE TABLE t1(a int, b int, c int not null) WITH(orientation=row);
CREATE TABLE t2(a int, b int, c int not null) WITH(orientation=row);
```

Run the following SQL statement to query the **NOT IN** execution plan:

```
EXPLAIN VERBOSE SELECT * FROM t1 WHERE t1.c NOT IN (SELECT t2.c FROM t2);
```

The following figure shows the statement output.



```
                                    QUERY PLAN
--------------------------------------------------------------------------------------
 id |                 operation                  | E-rows | E-distinct | E-width | E-costs
----+--------------------------------------------+--------+------------+---------+--------
  1 | ->  Streaming (type: GATHER)               |      6 |            |      12 | 78.98
  2 |    ->  Nested Loop Anti Join (3, 4)        |      6 |            |      12 | 64.98
  3 |       ->  Seq Scan on public.t1            |     60 |            |      12 | 18.18
  4 |       ->  Materialize                      |    360 |            |       4 | 30.75
  5 |          ->  Streaming(type: BROADCAST)    |    360 |            |       4 | 30.45
  6 |             ->  Seq Scan on public.t2      |     60 |            |       4 | 18.18

             Predicate Information (identified by plan id)
--------------------------------------------------------------------
  2 --Nested Loop Anti Join (3, 4)
        Join Filter: ((t1.c = t2.c) OR (t1.c IS NULL) OR (t2.c IS NULL))
```

According to the returned result, nest loops are used. As the OR operation result of NULL and any value is NULL,

```
t1.c NOT IN (SELECT t2.c FROM t2)
```

the preceding condition expression is equivalent to:

```
t1.c <> ANY(t2.c) AND t1.c IS NOT NULL AND ANY(t2.c) IS NOT NULL
```

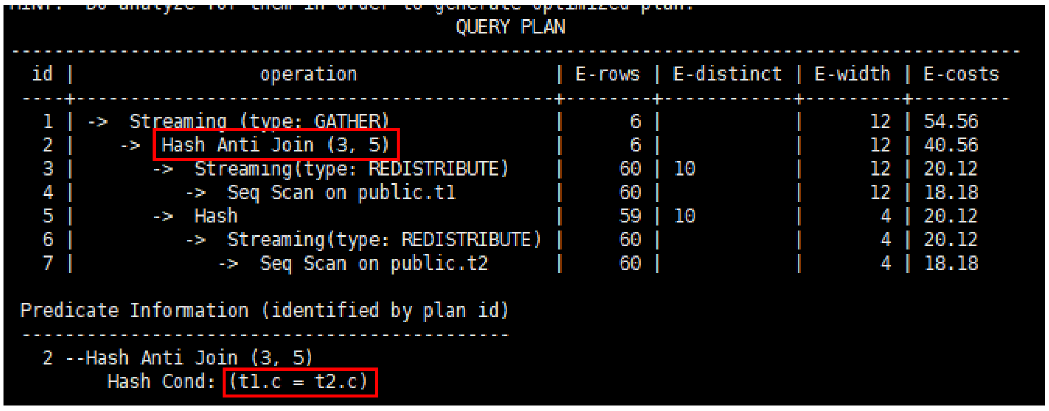
After Optimization

The query can be modified as follows:

```
SELECT * FROM t1 WHERE NOT EXISTS (SELECT * FROM t2 WHERE t2.c = t1.c);
```

Run the following statement to query the execution plan of **NOT EXISTS**:

```
EXPLAIN VERBOSE SELECT * FROM t1 WHERE NOT EXISTS (SELECT 1 FROM t2 WHERE t2.c = t1.c);
```



```
                                    QUERY PLAN
----------------------------------------------------------------------------------------
 id |                  operation                  | E-rows | E-distinct | E-width | E-costs
----+---------------------------------------------+--------+------------+---------+--------
  1 | ->  Streaming (type: GATHER)                |      6 |            |      12 | 54.56
  2 |    ->  Hash Anti Join (3, 5)                |      6 |            |      12 | 40.56
  3 |       ->  Streaming(type: REDISTRIBUTE)     |     60 | 10         |      12 | 20.12
  4 |          ->  Seq Scan on public.t1          |     60 |            |      12 | 18.18
  5 |       ->  Hash                              |     59 | 10         |       4 | 20.12
  6 |          ->  Streaming(type: REDISTRIBUTE)  |     60 |            |       4 | 20.12
  7 |             ->  Seq Scan on public.t2       |     60 |            |       4 | 18.18

 Predicate Information (identified by plan id)
---------------------------------------------
  2 --Hash Anti Join (3, 5)
        Hash Cond: (t1.c = t2.c)
```

14 GaussDB(DWS) System Catalogs and Views

14.1 Overview of System Catalogs and System Views

System catalogs are used by GaussDB(DWS) to store structure metadata. They are a core component the GaussDB(DWS) database system and provide control information for the database system. These system catalogs contain cluster installation information and information about various queries and processes in GaussDB(DWS). You can collect information about the database by querying the system catalog.

System views provide ways to query system catalogs and internal database status. 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 basic table. It is only a virtual object rather than a physical one. A database only stores the definition of a view and does not store its data. The data is still stored in the original 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.

In separation of duty, non-administrators have no permission to view system catalogs and views. In other scenarios, system catalogs and views are either visible only to administrators or visible to all users. System catalogs and views that require system administrator permissions can be queried only by system administrators.

NOTICE

- Do not add, delete, or modify system catalogs or system views. Manual modification or damage to system catalogs or system views may cause system information inconsistency, system control exceptions, or even cluster unavailability.
- System catalogs do not support toast and cannot be stored across pages. If the size of a page in a system catalog is 8 KB, the length of each field must be less than 8 KB.

Table 14-1 Common system catalogs

System Catalog	Description
PG_AM	Stores information about index access methods. There is one row for each index access method supported by the system.
PG_ATTRIBUTE	Stores information about table columns.
PG_AUTHID	Stores information about database authorization 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 the rolcanlogin is set or not, can use other roles as members. For a cluster, only one **pg_authid** exists which is not available for every database. It is accessible only to users with system administrator rights.
PG_CONSTRAINT	Stores check, primary key, unique, and foreign key constraints on tables.
PG_CLASS	Stores information about database objects and their relationships.
PG_DATABASE	Stores information about the available databases.
PG_DEPEND	Records dependencies among 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.
PG_PARTITION	Stores information about all partition tables (partitioned tables), partitions (table partitions), toast tables in partitions, and partition indexes (index partitions) in the database. Partitioned index information is not stored in the **PG_PARTITION** system catalog.
PG_FOREIGN_TABLE	Stores auxiliary information about foreign tables.
PG_INDEX	Stores part of the information about indexes. The rest is mostly stored in **PG_CLASS**.

System Catalog	Description
PG_JOBS	Stores detailed information about scheduled tasks created by users. The scheduled task threads periodically poll the **pg_jobs** system catalog and are automatically executed at the schedule time. This catalog belongs to the Shared Relation category. All job records are visible to all databases.
PG_LARGEOBJECT	Stores 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 to be LOBLKSIZE. It is accessible only to users with system administrator rights.
PG_NAMESPACE	Stores namespaces, which are schema-related information.
PG_PROC	Stores information about functions or procedures.

Table 14-2 Common system views

System View	Description
GS_CLUSTER_RESOURCE_INFO	Displays the DN resource summary.
GS_SQL_COUNT	Displays statistics about the five types of statements (**SELECT**, **INSERT**, **UPDATE**, **DELETE**, and **MERGE INTO**) executed on the current node of the database, including the number of execution times, response time (the maximum, minimum, average, and total response time of the other four types of statements except the **MERGE INTO** statement, in microseconds), and the number of execution times of **DDL**, **DML**, and **DCL statements**.
PG_LOCKS	Stores information about locks held by opened transactions.
PG_ROLES	Provides information about database access roles.
PG_RULES	Provides access to query useful information about rewrite rules.
PG_TOTAL_USER_RESOURCE_INFO	Displays resource usage of all users. Only administrators can query this view. This view is valid only when **se_workload_manager** is set to **on**.
PG_USER	Provides information about users who access the database.

System View	Description
PG_VIEWS	Provides useful information about access to each view in the database.
PG_STAT_ACTIVITY	Displays information about the current user's queries. If you have the rights of an administrator or the preset role, you can view all information about user queries.
PG_TABLES	Provides useful information about access to each table in the database.
PLAN_TABLE	Displays plan information collected by **EXPLAIN PLAN**. Plan information is in a session-level life cycle. After the session exits, the data will be deleted. Data is isolated between sessions and between users.

14.2 System Catalogs

14.2.1 GS_OBSSCANINFO

GS_OBSSCANINFO defines the OBS runtime information scanned in cluster acceleration scenarios. Each record corresponds to a piece of runtime information of a foreign table on OBS in a query.

Table 14-3 GS_OBSSCANINFO columns

Name	Type	Description
query_id	bigint	Specifies a query ID.
user_id	text	Specifies a database user who performs queries.
table_name	text	Specifies the name of a foreign table on OBS.
file_type	text	Specifies the format of files storing the underlying data.
time_stamp	time_stam	Specifies the scanning start time.
actual_time	double	Specifies the scanning execution time in seconds.
file_scanned	bigint	Specifies the number of files scanned.
data_size	double	Specifies the size of data scanned in bytes.
billing_info	text	Specifies the reserved fields.

14.2.2 GS_RESPOOL_RESOURCE_HISTORY

The **GS_RESPOOL_RESOURCE_HISTORY** table records the historical monitoring information about a resource pool on both CNs and DNs.

Table 14-4 GS_RESPOOL_RESOURCE_HISTORY columns

Name	Type	Description
timestamp	timestamp	Time when resource pool monitoring information is persistently stored
nodegroup	name	Name of the logical cluster of the resource pool. The default value is **installation**.
rpname	name	Resource pool name
cgroup	name	Name of the Cgroup associated with the resource pool
ref_count	int	Number of jobs referenced by the resource pool. The number is counted regardless of whether the job is controlled by the resource pool. This parameter is valid only on CNs.
fast_run	int	Number of running jobs in the fast lane of the resource pool. This parameter is valid only on CNs.
fast_wait	int	Number of jobs queued in the fast lane of the resource pool. This parameter is valid only on CNs.
fast_limit	int	Limit on the number of concurrent fast lane jobs in the resource pool. This parameter is valid only on CNs.
slow_run	int	Number of running jobs in the slow lane of the resource pool. This parameter is valid only on CNs.
slow_wait	int	Number of jobs queued in the slow lane of the resource pool. This parameter is valid only on CNs.
slow_limit	int	Limit on the number of concurrent slow lane jobs in the resource pool. This parameter is valid only on CNs.

Name	Type	Description
used_cpu	double	Average number of used CPUs of the resource pool in a 5s monitoring period. The value is accurate to two decimal places. ● On a DN, it indicates the number of CPUs used by the resource pool on the current DN. ● On a CN, it indicates the total CPU usage of resource pools on all DNs.
cpu_limit	int	It indicates the upper limit of available CPUs for resource pools. If the CPU time limit is specified, this parameter indicates the available CPUs for GaussDB(DWS). If the CPU usage limit is specified, this parameter indicates the available CPUs for associated Cgroups. ● On a DN, it indicates the upper limit of available CPUs for the resource pool on the current DN. ● On a CN, it indicates the total upper limit of available CPUs for resource pools on all DNs.
used_mem	int	Memory used by the resource pool, in MB. ● On a DN, it indicates the memory usage of the resource pool on the current DN. ● On a CN, it indicates the total memory usage of resource pools on all DNs.
estimate_mem	int	Estimated memory used by the jobs running in the resource pool on the current CN. This parameter is valid only on CNs.
mem_limit	int	Upper limit of available memory for resource pools, in MB. ● On a DN, it indicates the upper limit of available memory for the resource pool on the current DN. ● On a CN, it indicates the total upper limit of available memory for resource pools on all DNs.
read_kbytes	bigint	Number of logical read bytes in the resource pool within a 5s monitoring period (unit: KB). ● On a DN, it indicates the number of logical read bytes in the resource pool on the current DN. ● On a CN, it indicates the total logical read bytes of resource pools on all DNs.

Name	Type	Description
write_kbytes	bigint	Number of logical write bytes in the resource pool within a 5s monitoring period (unit: KB). ● On a DN, it indicates the number of logical write bytes in the resource pool on the current DN. ● On a CN, it indicates the total logical write bytes of resource pools on all DNs.
read_counts	bigint	Number of logical reads in the resource pool within a 5s monitoring period. ● On a DN, it indicates the number of logical reads in the resource pool on the current DN. ● On a CN, it indicates the total number of logical reads in resource pools on all DNs.
write_counts	bigint	Number of logical writes in the resource pool within a 5s monitoring period. ● On a DN, it indicates the number of logical writes in the resource pool on the current DN. ● On a CN, it indicates the total number of logical writes in resource pools on all DNs.
read_speed	double	Average rate of logical reads of the resource pool in a 5s monitoring period. ● On a DN, it indicates the logical read rate of the resource pool on the current DN. ● On a CN, it indicates the overall logical read rate of resource pools on all DNs.
write_speed	double	Average rate of logical writes of resource pools in a 5s monitoring period. ● On a DN, it indicates the logical write rate of resource pools on the current DN. ● On a CN, it indicates the overall logical write rate of resource pools on all DNs.

14.2.3 GS_WLM_INSTANCE_HISTORY

The **GS_WLM_INSTANCE_HISTORY** system catalog stores information about resource usage related to CN or DN instances. Each record in the system table indicates the 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.

Table 14-5 GS_WLM_INSTANCE_HISTORY column

Name	Type	Description
instancename	text	Instance name
timestamp	timestamp with time zone	Timestamp
used_cpu	int	CPU usage of an instance
free_mem	int	Unused memory of an instance (unit: MB)
used_mem	int	Used memory of an instance (unit: MB)
io_await	real	Specifies the **io_wait** value (average value within 10 seconds) of the disk used by an instance.
io_util	real	Specifies the **io_util** value (average value within 10 seconds) of the disk used by an instance.
disk_read	real	Specifies the disk read rate (average value within 10 seconds) of an instance (unit: KB/s).
disk_write	real	The disk write rate (average value within 10 seconds) of an instance (unit: KB/s).
process_read	bigint	Specifies the read rate (excluding the number of bytes read from the disk pagecache) of the corresponding instance process that reads data from a disk. (Unit: KB/s)
process_write	bigint	Specifies the 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. (Unit: KB/s)
logical_read	bigint	CN instance: N/A DN instance: Specifies the logical read byte rate of the instance in the statistical interval (10 seconds). (Unit: KB/s)
logical_write	bigint	CN instance: N/A DN instance: Specifies the logical write byte rate of the instance within the statistical interval (10 seconds). (Unit: KB/s)
read_counts	bigint	CN instance: N/A DN instance: Specifies the total number of logical read operations of the instance in the statistical interval (10 seconds).

Name	Type	Description
write_counts	bigint	CN instance: N/A DN instance: Specifies the total number of logical write operations of the instance in the statistical interval (10 seconds).

14.2.4 GS_WLM_OPERATOR_INFO

GS_WLM_OPERATOR_INFO records operators of completed jobs. The data is dumped from the kernel to a system catalog. If the GUC parameter **enable_resource_record** is set to **on**, the system imports records in **GS_WLM_OPERATOR_HISTORY** to this system catalog every three minutes. You are not advised to enable this function because it occupies storage space and affects performance.

NOTE

- This system catalog's schema is **dbms_om**.
- The **pg_catalog** has the **GS_WLM_OPERATOR_INFO** view.

Table 14-6 GS_WLM_OPERATOR_INFO columns

Name	Type	Description
nodename	text	Name of the CN where the statement is executed
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 plan_node_id
start_time	timestamp with time zone	Time when an operator starts to process the first data record
duration	bigint	Total execution time of an operator. The unit is ms.
query_dop	integer	Degree of parallelism (DOP) of the current operator
estimated_rows	bigint	Number of rows estimated by the optimizer
tuple_processed	bigint	Number of elements returned by the current operator

Name	Type	Description
min_peak_memory	integer	Minimum peak memory used by the current operator on all DNs. The unit is MB.
max_peak_memory	integer	Maximum peak memory used by the current operator on all DNs. The unit is MB.
average_peak_memory	integer	Average peak memory used by the current operator on all DNs. The unit is MB.
memory_skew_percent	integer	Memory usage skew of the current operator among DNs
min_spill_size	integer	Minimum spilled data among all DNs when a spill occurs. The unit is MB. Default value: **0**.
max_spill_size	integer	Maximum spilled data among all DNs when a spill occurs. The unit is MB. Default value: **0**.
average_spill_size	integer	Average spilled data among all DNs when a spill occurs. The unit is MB. Default value: **0**.
spill_skew_percent	integer	DN spill skew when a spill occurs
min_cpu_time	bigint	Minimum execution time of the operator on all DNs. The unit is ms.
max_cpu_time	bigint	Maximum execution time of the operator on all DNs. The unit is ms.
total_cpu_time	bigint	Total execution time of the operator on all DNs. The unit is ms.
cpu_skew_percent	integer	Skew of the execution time among DNs.
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

14.2.5 GS_WLM_SESSION_INFO

GS_WLM_SESSION_INFO records load management information about a completed job executed on all CNs. The data is dumped from the kernel to a system catalog. If the GUC parameter **enable_resource_record** is set to **on**, the system imports records in **GS_WLM_SESSION_HISTORY** to this system catalog every three minutes. You are not advised to enable this function because it occupies storage space and affects performance. For details about the columns, see **Table 14-137**.

NOTE

- This system catalog's schema is **dbms_om**.
- The **pg_catalog** has the **GS_WLM_SESSION_INFO** view.

14.2.6 GS_WLM_USER_RESOURCE_HISTORY

The **GS_WLM_USER_RESOURCE_HISTORY** system table stores information about resources used by users and is valid only on CNs. Each record in the system table indicates the 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 times, and logical I/O rate. The memory, CPU, and I/O monitoring items record only the resource usage of complex jobs.

Data in the **GS_WLM_USER_RESOURCE_HISTORY** system table comes from the **PG_TOTAL_USER_RESOURCE_INFO** view.

Table 14-7 GS_WLM_USER_RESOURCE_HISTORY column

Name	Type	Description
username	text	Username
timestamp	timestamp with time zone	Timestamp
used_memory	int	Memory size used by a user (MB). • DN: The memory used by users on the current DN is displayed. • CN: The total memory usage of users on all DNs is displayed.
total_memory	int	Memory used by the resource pool (MB). **0** indicates that the available memory is not limited and depends on the maximum memory available in the database (**max_dynamic_memory**). A calculation formula is as follows: total_memory = max_dynamic_memory * parent_percent * user_percent CN: The sum of maximum available memory on all DNs is displayed.
used_cpu	real	Number of CPU cores in use
total_cpu	int	Total number of CPU cores of the Cgroup associated with a user on the node

Name	Type	Description
used_space	bigint	Used storage space (KB)
total_space	bigint	Size of the storage space that can be used (KB). The value **-1** indicates that the maximum storage space is not limited.
used_temp_space	bigint	Used temporary storage space (KB)
total_temp_space	bigint	Available temporary storage space (KB). The value **-1** indicates that the maximum temporary storage space is not limited.
used_spill_space	bigint	Used space of operator flushing (KB)
total_spill_space	bigint	Available spill space (KB) The value **-1** indicates that the maximum spill space.
read_kbytes	bigint	Byte traffic of read operations in a monitoring period (KB)
write_kbytes	bigint	Byte traffic of write operations in a monitoring period (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 (KB)
write_speed	real	Byte rate of write operations in a monitoring period (KB)

14.2.7 PG_AGGREGATE

pg_aggregate records information about aggregation 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 14-8 PG_AGGREGATE columns

Name	Type	Reference	Description
aggfnoid	regproc	PG_PROC.oid	PG_PROC OID of the aggregate function

Name	Type	Reference	Description
aggtransfn	regproc	PG_PROC.oid	Transition function
aggcollectfn	regproc	PG_PROC.oid	Aggregate function
aggfinalfn	regproc	PG_PROC.oid	Final function (zero if none)
aggsortop	oid	PG_OPERATOR.oid	Associated sort operator (zero if none)
aggtranstype	oid	PG_TYPE.oid	Data type of the aggregate function's internal transition (state) data
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 out 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 out null.

14.2.8 PG_AM

PG_AM records information about index access methods. There is one row for each index access method supported by the system.

Table 14-9 PG_AM columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute; must be explicitly selected)
amname	name	-	Name of the access method
amstrategies	smallint	-	Number of operator strategies for this access method, or zero if access method does not have a fixed set of operator strategies
amsupport	smallint	-	Number of support routines for this access method

Name	Type	Reference	Description
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 a scan 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 an index storage data type can differ from a 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
amkeytype	oid	**PG_TYPE**.oid	Type of data stored in index, or zero if not a fixed type
aminsert	regproc	**PG_PROC**.oid	"Insert this tuple" function
ambeginscan	regproc	**PG_PROC**.oid	"Prepare for index scan" function
amgettuple	regproc	**PG_PROC**.oid	"Next valid tuple" function, or zero if none
amgetbitmap	regproc	**PG_PROC**.oid	"Fetch all valid tuples" function, or zero if none

Name	Type	Reference	Description
amrescan	regproc	**PG_PROC**.oid	"(Re)start index scan" function
amendscan	regproc	**PG_PROC**.oid	"Clean up after index scan" function
ammarkpos	regproc	**PG_PROC**.oid	"Mark current scan position" function
amrestrpos	regproc	**PG_PROC**.oid	"Restore marked scan position" function
ammerge	regproc	**PG_PROC**.oid	"Merge multiple indexes" function
ambuild	regproc	**PG_PROC**.oid	"Build new index" function
ambuildempty	regproc	**PG_PROC**.oid	"Build empty index" function
ambulkdelete	regproc	**PG_PROC**.oid	Bulk-delete function
amvacuumcleanup	regproc	**PG_PROC**.oid	Post-**VACUUM** cleanup function
amcanreturn	regproc	**PG_PROC**.oid	Function to check whether index supports index-only scans, or zero if none
amcostestimate	regproc	**PG_PROC**.oid	Function to estimate cost of an index scan
amoptions	regproc	**PG_PROC**.oid	Function to parse and validate **reloptions** for an index

14.2.9 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 14-10 PG_AMOP columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute; must be explicitly selected)
amopfamily	oid	**PG_OPFAMILY**.oid	Operator family this entry is for

Name	Type	Reference	Description
amoplefttype	oid	PG_TYPE.oid	Left-hand input data type of operator
amoprighttype	oid	PG_TYPE.oid	Right-hand input data type of operator
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	Index access method the operator family is for
amopsortfamily	oid	PG_OPFAMILY.oid	The btree operator family this entry sorts according to, if an ordering operator; zero if 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 the **ORDER BY** are specified by the **amopsortfamily** column, which must reference a btree operator family for the operator's result type.

14.2.10 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 belonging to an operator family.

Table 14-11 PG_AMPROC columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute; must be explicitly selected)
amprocfamily	oid	PG_OPFAMILY.oid	Operator family this entry is for

Name	Type	Reference	Description
amproclefttype	oid	**PG_TYPE**.oid	Left-hand input data type of associated operator
amprocrighttype	oid	**PG_TYPE**.oid	Right-hand input data type of associated operator
amprocnum	smallint	-	Support procedure number
amproc	regproc	**PG_PROC**.oid	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**.

14.2.11 PG_ATTRDEF

PG_ATTRDEF stores default values of columns.

Table 14-12 PG_ATTRDEF columns

Name	Type	Description
adrelid	oid	Table to which the column belongs
adnum	smallint	Number of a column.
adbin	pg_node_tree	Internal representation of the default value of the column
adsrc	text	Internal representation of the readable default value

14.2.12 PG_ATTRIBUTE

PG_ATTRIBUTE records information about table columns.

Table 14-13 PG_ATTRIBUTE columns

Name	Type	Description
attrelid	oid	Table to which the column belongs
attname	name	Column name
atttypid	oid	Column type

Name	Type	Description
attstattarget	integer	Controls the level of details of statistics collected for this column by **ANALYZE**. ● A zero value indicates that no statistics should be collected. ● A negative value says to use the system default statistics target. ● 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** of the column's type
attnum	smallint	Number of a column.
attndims	integer	Number of dimensions if the column is an array; otherwise, the value is 0.
attcacheoff	integer	This column is always -1 on disk. 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 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 the column's type
attstorage	"char"	Copy of **pg_type.typstorage** of this column's type
attalign	"char"	Copy of **pg_type.typalign** of the column's type
attnotnull	boolean	A not-null constraint. It is possible to change this column to enable or disable the constraint.
atthasdef	boolean	Indicates that 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.

Name	Type	Description
attisdropped	boolean	Whether the column has been dropped and is no longer valid. A dropped column is still physically present in the table but is ignored by the analyzer, so it cannot be accessed through SQL.
attislocal	boolean	Whether the column is defined locally in the relation. 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 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[]	Property-level options
attfdwoptions	text[]	Property-level external data options
attinitdefval	bytea	**attinitdefval** stores the default value expression. **ADD COLUMN** in a row-store table must use this column.
attkvtype	tinyint	**kv_type** attribute of a column. Values: ● **0** indicates the default value, which is used for non-time series tables. ● **1** indicates TSTAG, a dimension attribute, which is used only for time series tables. ● **2** indicates TSFIELD, a metric attribute, which is used only for the time sequence table. ● **3** indicates TSTIME, a time attribute, which is used only for time series tables.

Example

Query the field names and field IDs of a specified table. Replace **t1** and **public** with the actual table name and schema name, respectively.

```
SELECT attname,attnum FROM pg_attribute WHERE attrelid=(SELECT pg_class.oid FROM pg_class JOIN
pg_namespace ON relnamespace=pg_namespace.oid WHERE relname='t1' and nspname='public') and
attnum>0;
     attname      | attnum
------------------+--------
 product_id       |      1
 product_name     |      2
 product_quantity |      3
(3 rows)
```

14.2.13 PG_AUTHID

PG_AUTHID records information about the 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 the rolcanlogin is set or not, can use other roles as members.

For a cluster, only one **pg_authid** exists which is not available for every database. It is accessible only to users with system administrator rights.

Table 14-14 PG_AUTHID columns

Column	Type	Description
oid	oid	Row identifier (hidden attribute; must be explicitly selected)
rolname	name	Role name
rolsuper	boolean	Whether the role is the initial system administrator with the highest permission
rolinherit	boolean	Whether the role automatically inherits permissions of roles it is a member of
rolcreaterole	boolean	Whether the role can create more roles
rolcreatedb	boolean	Whether the role can create databases
rolcatupdate	boolean	Whether the role can directly update system catalogs. Only the initial system administrator whose usesysid is 10 has this permission. It is not available for other users.
rolcanlogin	boolean	Whether a role can log in, that is, whether a role can be given as the initial session authorization identifier.
rolreplication	boolean	Indicates that the role is a replicated one (an adaptation syntax and no actual meaning).
rolauditadmin	boolean	Indicates that the role is an audit user.
rolsystemadmin	boolean	Indicates that the role is an administrator.
rolconnlimit	integer	Limits the maximum number of concurrent connections of a user on a CN node. -1 means no limit.

Column	Type	Description
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
rolparentid	oid	OID of a group user to which the user belongs
roltabspace	Text	Storage space of the user permanent table
rolkind	char	Special type of user, including private users, logical cluster administrators, and common users.
rolnodegroup	oid	OID of a node group associated with a user. The node group must be a logical cluster.
roltempspace	Text	Storage space of the user temporary table
rolspillspace	Text	Operator disk spill space of the user
rolexcpdata	text	Reserved column
rolauthinfo	text	Additional information when LDAP authentication is used. If other authentication modes are used, the value is **NULL**.
rolpwdexpire	integer	Password expiration time. Users can change their password before it expires. After the password expires, only the administrator can change the password. The value **-1** indicates that the password never expires.
rolpwdtime	timestamp with time zone	Time when a password is created

14.2.14 PG_AUTH_HISTORY

PG_AUTH_HISTORY records the authentication history of the role. It is accessible only to users with system administrator rights.

Table 14-15 PG_AUTH_HISTORY columns

Name	Type	Description
roloid	oid	ID of the role
passwordtime	timestamp with time zone	Time of password creation and change
rolpassword	text	Role password that is encrypted using MD5 or SHA256, or that is not encrypted

14.2.15 PG_AUTH_MEMBERS

PG_AUTH_MEMBERS records the membership relations between roles.

Table 14-16 PG_AUTH_MEMBERS columns

Name	Type	Description
roleid	oid	ID of a role that has a member
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

14.2.16 PG_CAST

PG_CAST records conversion relationships between data types.

Table 14-17 PG_CAST columns

Name	Type	Description
castsource	oid	OID of the source data type
casttarget	oid	OID of the target data type
castfunc	oid	OID of the conversion function. If the value is **0**, no conversion function is required.

Name	Type	Description
castcontext	"char"	Conversion mode between the source and target data types ● **e** indicates that only explicit conversion can be performed (using the CAST or :: syntax). ● **i** indicates that only implicit conversion can be performed. ● **a** indicates that both explicit and implicit conversion can be performed between data types.
castmethod	"char"	Conversion method ● **f** indicates that conversion is performed using the specified function in the **castfunc** column. ● **b** indicates that binary forcible conversion rather than the specified function in the **castfunc** column is performed between data types.

14.2.17 PG_CLASS

PG_CLASS records database objects and their relations.

Table 14-18 PG_CLASS columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute; must be explicitly selected)
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 this table's row type (the index is 0 because the index does not have **pg_type** record)
reloftype	oid	OID is of composite type. **0** indicates other types.
relowner	oid	Owner of the relationship
relam	oid	Specifies the 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. If such file does not exist, the value is **0**.

Name	Type	Description
reltablespace	oid	Tablespace in which this relationship is stored. If its 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 this 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 this table. The OID is 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. The OID is 0 for a table other than a TOAST table.
reldeltarelid	oid	OID of a Delta table Delta tables belong to column-store tables. They store long tail data generated during data import.
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 databases in the cluster. Only certain system catalogs (such as **pg_database**) are shared.

Name	Type	Description
relpersistence	"char"	● **p** indicates a permanent table. ● **u** indicates a non-log table. ● **t** indicates a temporary table.
relkind	"char"	● **r** indicates an ordinary table. ● **i** indicates an index. ● **S** indicates a sequence. ● **v** indicates a view. ● **c** indicates the composite type. ● **t** indicates a TOAST table. ● **f** indicates a foreign table.
relnatts	smallint	Number of user columns in the relationship (excluding system columns) **pg_attribute** has the same number of rows corresponding to the user columns.
relchecks	smallint	Number of constraints on a table. For details, see **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.
relhasrules	boolean	Its value is **true** if the table has rules. See table **PG_REWRITE** to check whether it has rules.
relhastriggers	boolean	Its value is **true** if the table has (or once had) triggers. For details, see **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** indicates other tables that do not support compression (primarily system tables, on which the compression attribute cannot be modified). ● **1** indicates that the compression feature of the table data is NOCOMPRESS or has no specified keyword. ● **2** indicates that the compression feature of the table data is COMPRESS.
relhasclusterkey	boolean	Whether the local cluster storage is used

Name	Type	Description
relrowmovement	boolean	Whether the row migration is allowed when the partitioned table is updated ● **true** indicates that the row migration is allowed. ● **false** indicates that the row migration is not allowed.
parttype	"char"	Whether the table or index has the property of a partitioned table ● **p** indicates that the table or index has the property of a partitioned table. ● **n** indicates that the table or index does not have the property of a partitioned table. ● **v** indicates that the table is the value partitioned table in the HDFS.
relfrozenxid	xid32	All transaction IDs before this one have been replaced with a permanent ("frozen") transaction ID in this table. This column is used to track whether the table needs to be vacuumed in order 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.

relacl	aclitem[]	Access permissions
		The command output of the query is as follows:
		`rolename=xxxx/yyyy  --Assigning privileges to a role`
		`=xxxx/yyyy --Assigning the permission to public`
		xxxx indicates the assigned privileges, and *yyyy* indicates the roles that are assigned to the privileges. For details about permission descriptions, see **Table 14-19**.

reloptions	text[]	Access-method-specific options, as "keyword=value" strings
relfrozenxid64	xid	All transaction IDs before this one have been replaced with a permanent ("frozen") transaction ID in this table. This column is used to track whether the table needs to be vacuumed in order 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.

Table 14-19 Description of privileges

Parameter	Description
r	SELECT (read)
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	ANALYZE|ANALYSE
L	ALTER
P	DROP
v	VACUUM
arwdDxtA, vLP	ALL PRIVILEGES (used for tables)
*	Authorization options for preceding permissions

Examples

View the OID and relfilenode of a table.

```
SELECT oid,relname,relfilenode FROM pg_class WHERE relname = 'table_name';
```

Count row-store tables.

```
SELECT 'row count:'||count(1) as point FROM pg_class WHERE relkind = 'r' and oid > 16384 and
reloptions::text not like '%column%' and reloptions::text not like '%internal_mask%';
```

Count column-store tables.

```
SELECT 'column count:'||count(1) as point FROM pg_class WHERE relkind = 'r' and oid > 16384 and
reloptions::text like '%column%';
```

Query the comments of all tables in the database:

```
SELECT relname as tabname,obj_description(relfilenode,'pg_class') as comment FROM pg_class;
```

14.2.18 PG_COLLATION

PG_COLLATION records the available collations, which are essentially mappings from an SQL name to operating system locale categories.

Table 14-20 PG_COLLATION columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute; must be explicitly selected)
collname	name	-	Collation name (unique per namespace and encoding)
collnamespace	oid	**PG_NAMESPACE**.oid	OID of the namespace that contains this collation
collowner	oid	**PG_AUTHID**.oid	Owner of the collation
collencoding	integer	-	Encoding in which the collation is applicable, or **-1** if it works for any encoding **NOTE** You can use the **pg_encoding_to_char()** function to convert a number to the corresponding code name.
collcollate	name	-	**LC_COLLATE** for this collation object
collctype	name	-	**LC_CTYPE** for this collation object

14.2.19 PG_CONSTRAINT

PG_CONSTRAINT records check, primary key, unique, and foreign key constraints on the tables.

Table 14-21 PG_CONSTRAINT columns

Name	Type	Description
conname	name	Constraint name (not necessarily unique)
connamespace	oid	OID of the namespace that contains the constraint

Name	Type	Description
contype	"char"	● **c** indicates check constraints. ● **f** indicates foreign key constraints. ● **p** indicates primary key constraints. ● **u** indicates unique constraints. ● **t** indicates trigger constraints.
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, only foreign key and check constraints can be set to false.
conrelid	oid	Table containing this constraint. The value is **0** if it is not a table constraint.
contypid	oid	Domain containing this constraint. The value is **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** indicates no action. ● **r** indicates restriction. ● **c** indicates cascading. ● **n** indicates that the parameter is set to null. ● **d** indicates that the default value is used.
confdeltype	"char"	Foreign key deletion action code ● **a** indicates no action. ● **r** indicates restriction. ● **c** indicates cascading. ● **n** indicates that the parameter is set to null. ● **d** indicates that the default value is used.
confmatchtype	"char"	Foreign key match type ● **f** indicates full match. ● **p** indicates partial match. ● **u** indicates simple match (not specified).

Name	Type	Description
conislocal	boolean	Whether the local constraint is defined for the relationship
coninhcount	integer	Number of direct inheritance parent tables this constraint has. When the number 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 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
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
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

NOTICE

- **consrc** is not updated when referenced objects change; for example, it will not track renaming of columns. Rather than relying on this field, it is best to use **pg_get_constraintdef()** to extract the definition of a check constraint.
- **pg_class.relchecks** must be consistent with the number of check-constraint entries in this table for each relationship.

Example

Query whether a specified table has a primary key.

```
CREATE TABLE t1
(
    C_CUSTKEY     BIGINT        ,
    C_NAME        VARCHAR(25)   ,
    C_ADDRESS     VARCHAR(40)   ,
    C_NATIONKEY   INT           ,
    C_PHONE       CHAR(15)      ,
    C_ACCTBAL     DECIMAL(15,2),
    CONSTRAINT C_CUSTKEY_KEY PRIMARY KEY(C_CUSTKEY,C_NAME)
)
DISTRIBUTE BY HASH(C_CUSTKEY,C_NAME);

SELECT conname FROM pg_constraint WHERE conrelid = 't1'::regclass AND contype = 'p';
    conname
---------------
 c_custkey_key
(1 row)
```

14.2.20 PG_CONVERSION

PG_CONVERSION records encoding conversion information.

Table 14-22 PG_CONVERSION columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute; must be explicitly selected)
conname	name	-	Conversion name (unique in a namespace)
connamespace	oid	**PG_NAMESPACE**.oid	OID of the namespace that contains this conversion
conowner	oid	**PG_AUTHID**.oid	Owner of the conversion
conforencoding	integer	-	Source encoding ID
contoencoding	integer	-	Destination encoding ID
conproc	regproc	**PG_PROC**.oid	Conversion procedure
condefault	boolean	-	Its value is **true** if this is the default conversion.

14.2.21 PG_DATABASE

PG_DATABASE records information about the available databases.

Table 14-23 PG_DATABASE columns

Name	Type	Description
datname	name	Database name

Name	Type	Description
datdba	oid	Owner of the database, usually the user who created it
encoding	integer	Character encoding for this database You can use pg_encoding_to_char() to convert this number to the encoding name.
datcollate	name	Sequence used by the database
datctype	name	Character type used by the database
datistemplate	boolean	Whether this column can serve as a template database
datallowconn	boolean	If false then 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. **-1** indicates no limit.
datlastsysoid	oid	Last system OID in the database
datfrozenxid	xid32	Tracks whether the database needs to be vacuumed in order to prevent transaction ID wraparound. 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 ● **ORA**: compatible with the Oracle database ● **TD**: compatible with the Teradata database ● **MySQL**: compatible with the MySQL database
datacl	aclitem[]	Access permissions
datfrozenxid64	xid	Tracks whether the database needs to be vacuumed in order to prevent transaction ID wraparound.

Example

Run the following command to view the owner, compatibility mode, and access permissions of a database:

```
SELECT datname, datdba,datcompatibility,datacl from pg_database where datname='database_name';
```

View the encoding of a database:

```
SELECT pg_encoding_to_char(encoding) FROM pg_database WHERE datname='database_name';
```

14.2.22 PG_DB_ROLE_SETTING

PG_DB_ROLE_SETTING records the default values of configuration items bonded to each role and database when the database is running.

Table 14-24 PG_DB_ROLE_SETTING columns

Name	Type	Description
setdatabase	oid	Database corresponding to the configuration items; the value is **0** if the database is not specified
setrole	oid	Role corresponding to the configuration items; the value is **0** if the role is not specified
setconfig	text[]	Default value of configuration items when the database is running

14.2.23 PG_DEFAULT_ACL

PG_DEFAULT_ACL records the initial privileges assigned to the newly created objects.

Table 14-25 PG_DEFAULT_ACL columns

Name	Type	Description
defaclrole	oid	ID of the role associated with the permission
defaclnamespace	oid	Namespace associated with the permission; the value is **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.
defaclacl	aclitem[]	Access permissions that this type of object should have on creation

Examples

Run the following command to view the initial permissions of the new user **role1**:

```
select * from PG_DEFAULT_ACL;
 defaclrole | defaclnamespace | defaclobjtype |    defaclacl
------------+-----------------+---------------+-----------------
      16820 |           16822 | r             | {role1=r/user1}
```

You can also run the following statement to convert the format:

```
SELECT pg_catalog.pg_get_userbyid(d.defaclrole) AS "Granter",  n.nspname AS "Schema",  CASE
d.defaclobjtype WHEN 'r' THEN 'table' WHEN 'S' THEN 'sequence' WHEN 'f' THEN 'function' WHEN 'T'
THEN 'type' END AS "Type",  pg_catalog.array_to_string(d.defaclacl, E', ') AS "Access privileges" FROM
pg_catalog.pg_default_acl d LEFT JOIN pg_catalog.pg_namespace n ON n.oid = d.defaclnamespace ORDER
BY 1, 2, 3;
```

If the following information is displayed, **user1** grants **role1** the read permission on schema **user1**.

```
 Granter | Schema | Type  | Access privileges
---------+--------+-------+-------------------
 user1   | user1  | table | role1=r/user1
(1 row)
```

14.2.24 PG_DEPEND

PG_DEPEND records the dependency relationships 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 provides a similar function for dependencies involving objects that are shared across a database cluster.

Table 14-26 PG_DEPEND columns

Name	Type	Reference	Description
classid	oid	**PG_CLASS**.oid	OID of the system catalog the dependent object is in
objid	oid	Any OID column	OID of the specific dependent object
objsubid	integer	-	For a table column, this is the column number (the objid and classid refer to the table itself). For all other object types, this column is **0**.
refclassid	oid	**PG_CLASS**.oid	OID of the system catalog the referenced object is in
refobjid	oid	Any OID column	OID of the specific referenced object
refobjsubid	integer	-	For a table column, this is the column number (the refobjid and refclassid refer to the table itself). For all other object types, this column is **0**.
deptype	"char"	-	A code defining the specific semantics of this dependency relationship

In all cases, a **pg_depend** entry indicates that the referenced object cannot be dropped without also dropping the dependent object. However, there are several subflavors defined 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. Example: A trigger created to enforce a foreign-key constraint is made internally dependent on the constraint's **PG_CONSTRAINT** entry.
- DEPENDENCY_EXTENSION (e): dependent objects depended object extension of a member. For details, see **PG_EXTENSION**). The dependent object can be dropped 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 indicates that the system itself depends on the referenced object, and therefore the object cannot be deleted. Entries of this type are created only by **initdb**. The columns with dependent object are all zeroes.

Examples

Query the table that depends on the database object sequence **serial1**:

1. Query the OID of the sequence **serial1** in the system catalog **PG_CLASS**.
```
SELECT oid FROM pg_class WHERE relname ='serial1';
  oid
-------
 17815
(1 row)
```
2. Use the system catalog **PG_DEPEND** and the OID of **serial1** to obtain the objects that depend on **serial1**.
```
SELECT * FROM pg_depend WHERE objid ='17815';
 classid | objid | objsubid | refclassid | refobjid | refobjsubid | deptype
---------+-------+----------+------------+----------+-------------+---------
    1259 | 17815 |        0 |       2615 |     2200 |           0 | n
    1259 | 17815 |        0 |       1259 |    17812 |           1 | a
(2 rows)
```
3. Obtain the OID of the table that depends on the serial1 sequence based on the refobjid field and query the table name. The result indicates that the table **customer_address** depends on **serial1**.
```
SELECT relname FROM pg_class where oid='17812';
     relname
```

```
------------------
 customer_address
(1 row)
```

14.2.25 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 performs a similar function for descriptions involving objects that are shared across a database cluster.

Table 14-27 PG_DESCRIPTION columns

Name	Type	Reference	Description
objoid	oid	Any OID column	OID of the object this description pertains to
classoid	oid	**PG_CLASS**oid	OID of the system catalog this object appears in
objsubid	integer	-	For a comment on a table column, this is the column number (the **objoid** and **classoid** refer to the table itself). For all other object types, this column is **0**.
description	text	-	Arbitrary text that serves as the description of this object

14.2.26 PG_ENUM

PG_ENUM records entries showing the values and labels for each enum type. The internal representation of a given enum value is actually the OID of its associated row in **pg_enum**.

Table 14-28 PG_ENUM columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute; must be explicitly selected)
enumtypid	oid	**PG_TYPE**.oid	OID of the **pg_type** entry that contains this enum value
enumsortorder	real	-	Sort position of this enum value within its enum type
enumlabel	name	-	Textual label for this enum 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 enum type.

That is, if two even OIDs belong to the same enum 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 enum comparison routines to avoid catalog lookups in many common cases. The routines that create and alter enum types attempt to assign even OIDs to enum values whenever possible.

When an enum type is created, its members are assigned sort-order positions from 1 to *n*. But 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 enum type.

14.2.27 PG_EXTENSION

PG_EXTENSION records information about the installed extensions. By default, GaussDB(DWS) has 14 extensions: PLPGSQL, DIST_FDW, FILE_FDW, ROACH_API, HDFS_FDW, BTREE_GIN, GC_FDW, LOG_FDW, HSTORE, PACKAGES, PLDBGAPI, TSDB, DIMSEARCH, and UUID-OSSP.

Table 14-29 PG_EXTENSION

Name	Type	Description
extname	name	Extension name
extowner	oid	Owner of the extension
extnamespace	oid	Namespace containing the extension's exported objects
extrelocatable	boolean	Its value is **true** if the extension can be relocated to another schema.
extversion	text	Version number of the extension
extconfig	oid[]	Configuration information about the extension
extcondition	text[]	Filter conditions for the extension's configuration information

14.2.28 PG_EXTENSION_DATA_SOURCE

PG_EXTENSION_DATA_SOURCE records information about external data source. An external data source contains information about an external database, such as its password encoding. It is mainy used with Extension Connector.

Table 14-30 PG_EXTENSION_DATA_SOURCE columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute; must be explicitly selected)

Name	Type	Reference	Description
srcname	name	-	Name of an external data source
srcowner	oid	PG_AUTHID.oid	Owner of an external data source
srctype	text	-	Type of an external data source. It is NULL by default.
srcversion	text	-	Type of an external data source. It is NULL by default.
srcacl	aclitem[]	-	Access permissions
srcoptions	text[]	-	Option used for foreign data sources. It is a keyword=value string.

14.2.29 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 14-31 PG_FOREIGN_DATA_WRAPPER columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute; must be explicitly selected)
fdwname	name	-	Name of the foreign-data wrapper
fdwowner	oid	**PG_AUTHID**.oid	Owner of the foreign-data wrapper
fdwhandler	oid	**PG_PROC**.oid	References a handler function that is responsible for supplying execution routines for the foreign-data wrapper. Its value is **0** if no handler is provided.
fdwvalidator	oid	**PG_PROC**.oid	References a validator function that is responsible for checking the validity of the options given to the foreign-data wrapper, as well as options for foreign servers and user mappings using the foreign-data wrapper. Its value is **0** if no validator is provided.
fdwacl	aclitem[]	-	Access permissions

Name	Type	Reference	Description
fdwoptions	text[]	-	Option used for foreign data wrappers. It is a keyword=value string.

14.2.30 PG_FOREIGN_SERVER

PG_FOREIGN_SERVER records the 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 14-32 PG_FOREIGN_SERVER columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute; must be explicitly selected)
srvname	name	-	Name of the foreign server
srvowner	oid	**PG_AUTHID**.oid	Owner of the foreign server
srvfdw	oid	**PG_FOREIGN_DATA_WRAPPER**.oid	OID of the foreign-data wrapper of this foreign server
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. It is a keyword=value string.

14.2.31 PG_FOREIGN_TABLE

PG_FOREIGN_TABLE records auxiliary information about foreign tables.

Table 14-33 PG_FOREIGN_TABLE columns

Name	Type	Description
ftrelid	oid	OID of the foreign table
ftserver	oid	OID of the server where the foreign table is located

Name	Type	Description
ftwriteonly	boolean	Whether data can be written in the foreign table
ftoptions	text[]	Foreign table option

14.2.32 PG_INDEX

PG_INDEX records part of the information about indexes. The rest is mostly in **PG_CLASS**.

Table 14-34 PG_INDEX columns

Name	Type	Description
indexrelid	oid	OID of the **pg_class** entry for this index
indrelid	oid	OID of the **pg_class** entry for the table this index is for
indnatts	smallint	Number of columns in an index
indisunique	boolean	This index is a unique index if the value is **true**.
indisprimary	boolean	This index represents the primary key of the table if the value is **true**. If this value is **true**, the value of **indisunique** is true.
indisexclusion	boolean	This index supports exclusion constraints if the value is **true**.
indimmediate	boolean	A uniqueness check is performed upon data insertion if the value is **true**.
indisclustered	boolean	The table was last clustered on this index if the value is **true**.
indisusable	boolean	This index supports insert/select if the value is **true**.
indisvalid	boolean	This index is valid for queries if the value is **true**. If this column is **false**, this 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.

Name	Type	Description
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** event horizon, because the table may contain broken HOT chains with incompatible rows that they can see.
indisready	boolean	If the value is **true**, this index is ready for inserts. If the value is **false**, this index is ignored when data is inserted or modified.
indkey	int2vector	This is an array of **indnatts** values that indicate which table columns this index creates. For example, a value of **1 3** means that the first and the third columns make up the index key. **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 column contains the OID of the operator class to use. For details, see **PG_OPCLASS**.
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**. 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, the value is null.

14.2.33 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 14-35 PG_INHERITS columns

Name	Type	Reference	Description
inhrelid	oid	**PG_CLASS**.oid	OID of the child table
inhparent	oid	**PG_CLASS**.oid	OID of the 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.

14.2.34 PG_JOBS

PG_JOBS records detailed information about jobs created by users. Dedicated threads poll the **pg_jobs** table and trigger jobs based on scheduled job execution time. This table belongs to the Shared Relation category. All job records are visible to all databases.

Table 14-36 PG_JOBS columns

Name	Type	Description
job_id	integer	Job ID, primary key, unique (with a unique index)
what	text	Job content
log_user	oid	Username of the job creator
priv_user	oid	User ID of the job executor
job_db	oid	OID of the database where the job is executed
job_nsp	oid	OID of the namespace where a job is running
job_node	oid	CN node on which the job will be created and executed
is_broken	boolean	Indicates whether the current job is invalid.
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

Name	Type	Description
failure_count	smallint	Number of consecutive failures.
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
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

14.2.35 PG_LANGUAGE

PG_LANGUAGE records languages that can be used to write functions or stored procedures.

Table 14-37 PG_LANGUAGE columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute; must be explicitly selected)
lanname	name	-	Name of the language
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 to determine which languages need to be dumped, but this might be replaced by a different mechanism in the future.

Name	Type	Reference	Description
lanpltrusted	boolean	-	Its 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 external languages, this references the language handler, which is a special function that is responsible for executing all functions that are written in the particular language.
laninline	oid	**PG_PROC**.oid	This references a function that is 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 references a language validator function that is 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

14.2.36 PG_LARGEOBJECT

PG_LARGEOBJECT records the 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 to be LOBLKSIZE (which is currently BLCKSZ/4, or typically 2 kB).

It is accessible only to users with system administrator rights.

Table 14-38 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)

Name	Type	Reference	Description
data	bytea	-	Actual 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 are read as zeroes.

14.2.37 PG_LARGEOBJECT_METADATA

PG_LARGEOBJECT_METADATA records metadata associated with large objects. The actual large object data is stored in **PG_LARGEOBJECT**.

Table 14-39 PG_LARGEOBJECT_METADATA columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute; must be explicitly selected)
lomowner	oid	PG_AUTHID.oid	Owner of the large object
lomacl	aclitem[]	-	Access permissions

14.2.38 PG_NAMESPACE

PG_NAMESPACE records the namespaces, that is, schema-related information.

Table 14-40 PG_NAMESPACE columns

Name	Type	Description
nspname	name	Name of the namespace
nspowner	oid	Owner of the namespace
nsptimeline	bigint	Timeline when the namespace is created on the DN This column is for internal use and valid only on the DN.
nspacl	aclitem[]	Access permissions For details, see GRANT and REVOKE.
permspace	bigint	Quota of a schema's permanent tablespace
usedspace	bigint	Used size of a schema's permanent tablespace

14.2.39 PG_OBJECT

PG_OBJECT records the user creation, creation time, last modification time, and last analyzing time of objects of specified types (types existing in **object_type**).

Table 14-41 PG_OBJECT columns

Name	Type	Description
object_oid	oid	Object identifier.
object_type	"char"	Object type: ● **r** indicates a table, which can be an ordinary table or a temporary table. ● **i** indicates an index. ● **s** indicates a sequence. ● **v** indicates a view. ● **p** indicates a stored procedure and function.
creator	oid	ID of the creator.
ctime	timestamp with time zone	Object creation time.
mtime	timestamp with time zone	Time when the object was last modified. By default, the **ALTER**, **COMMENT**, **GRANT/ REVOKE**, and **TRUNCATE** operations are recorded.
last_analyze_time	timestamp with time zone	Time when an object is analyzed for the last time.

NOTICE

- Only normal user operations are recorded. Operations before the object upgrade and during the **initdb** process cannot be recorded.
- **ctime** and **mtime** are the start time of the transaction.
- The time of object modification due to capacity expansion is also recorded.

14.2.40 PG_OBSSCANINFO

PG_OBSSCANINFO defines the OBS runtime information scanned in cluster acceleration scenarios. Each record corresponds to a piece of runtime information of a foreign table on OBS in a query.

Table 14-42 PG_OBSSCANINFO columns

Name	Type	Reference	Description
query_id	bigint	-	Query ID
user_id	text	-	Database user who performs queries
table_name	text	-	Name of a foreign table on OBS
file_type	text	-	Format of files storing the underlying data
time_stamp	time_stam	-	Scanning start time
actual_time	double	-	Scanning execution time, in seconds
file_scanned	bigint	-	Number of files scanned
data_size	double	-	Size of data scanned, in bytes
billing_info	text	-	Reserved columns

14.2.41 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 lefthand input.

Table 14-43 PG_OPCLASS columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute; must be explicitly selected)
opcmethod	oid	**PG_AM**.oid	Index access method the operator class is for
opcname	name	-	Name of the operator class
opcnamespace	oid	**PG_NAMESPACE**.oid	Namespace to which the operator class belongs
opcowner	oid	**PG_AUTHID**.oid	Owner of the operator class
opcfamily	oid	**PG_OPFAMILY**.oid	Operator family containing the operator class
opcintype	oid	**PG_TYPE**.oid	Data type that the operator class indexes

Name	Type	Reference	Description
opcdefault	boolean	-	Whether the operator class is the default for **opcintype**. If it is, its value is **true**.
opckeytype	oid	**PG_TYPE**.oid	Type of data stored in index, or zero if same as **opcintype**

An operator class's **opcmethod** must match the **opfmethod** of its containing operator family. Also, there must be no more than one **pg_opclass** row having **opcdefault** true for any given combination of **opcmethod** and **opcintype**.

14.2.42 PG_OPERATOR

PG_OPERATOR records information about operators.

Table 14-44 PG_OPERATOR columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute; must be explicitly selected)
oprname	name	-	Name of the operator
oprnamespace	oid	**PG_NAMESPACE**.oid	OID of the namespace that contains this operator
oprowner	oid	**PG_AUTHID**.oid	Owner of the operator
oprkind	"char"	-	● **b**: infix ("both") ● **l**: prefix ("left") ● **r**: postfix ("right")
oprcanmerge	boolean	-	Whether the operator supports merge joins
oprcanhash	boolean	-	Whether the operator supports hash joins
oprleft	oid	**PG_TYPE**.oid	Type of the left operand
oprright	oid	**PG_TYPE**.oid	Type of the right operand
oprresult	oid	**PG_TYPE**.oid	Type of the result
oprcom	oid	**PG_OPERATOR**.oid	Commutator of this operator, if any
oprnegate	oid	**PG_OPERATOR**.oid	Negator of this operator, if any

Name	Type	Reference	Description
oprcode	regproc	PG_PROC.oid	Function that implements this operator
oprrest	regproc	PG_PROC.oid	Restriction selectivity estimation function for this operator
oprjoin	regproc	PG_PROC.oid	Join selectivity estimation function for this operator

14.2.43 PG_OPFAMILY

PG_OPFAMILY defines operator families.

Each operator family is a collection of operators and associated support routines that implement the 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 concept allows cross-data-type operators to be used with indexes and to be reasoned about using knowledge of access method semantics.

Table 14-45 PG_OPFAMILY columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute; must be explicitly selected)
opfmethod	oid	PG_AM.oid	Index access method the operator family is for
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 **PG_OPFAMILY**, but in the associated PG_AMOP, PG_AMPROC, and PG_OPCLASS.

14.2.44 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 **PG_PARTITION** system catalog.

Table 14-46 PG_PARTITION columns

Name	Type	Description
relname	name	Names of the partitioned tables, table partitions, TOAST tables on table partitions, and index partitions
parttype	"char"	Object type ● **r** indicates a partitioned table. ● **p** indicates a table partition. ● **x** indicates an index partition. ● **t** indicates a 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 field.
intervalnum	integer	Reserved field.
partstrategy	"char"	Partition policy of the partitioned table. The following policies are supported: **r** indicates the range partition. **v** indicates the numeric partition. **l**: indicates the list 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, TOAST table on the table partition
relpages	double precision	Statistics: numbers of data pages of the table partition and index partition
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

Name	Type	Description
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 a keyword=value string.
relfrozenxid64	xid	Frozen transaction ID

boundexprs	pg_node_tree	Partition boundary expression.
		● For range partitioning, it is the upper boundary expression of a partition.
		● For list partitioning, it is a collection of partition boundary enumeration values.
		The **pg_node_tree** data is not readable. You can use the expression **pg_get_expr** to translate the current column into readable information.
		`SELECT pg_get_expr(boundexprs, 0) FROM pg_partition WHERE relname = 'country_202201';`
		`pg_get_expr`
		`----------------------------------------------------------------`
		`ROW(202201, 'city1'::text), ROW(202201, 'city2'::text)`
		`(1 row)`

Example

Query the partition information of the partitioned table **web_returns_p2**.

```
CREATE TABLE web_returns_p2
(
    wr_returned_date_sk       integer,
    wr_returned_time_sk       integer,
    wr_item_sk                integer NOT NULL,
    wr_refunded_customer_sk   integer
)
WITH (orientation = column)
DISTRIBUTE BY HASH (wr_item_sk)
PARTITION BY RANGE(wr_returned_date_sk)
(
    PARTITION p2016 START(20161231) END(20191231) EVERY(10000),
    PARTITION p0 END(maxvalue)
);

SELECT oid FROM pg_class WHERE relname ='web_returns_p2';
  oid
-------
 97628

SELECT relname,parttype,parentid,boundaries FROM pg_partition WHERE parentid = '97628';
    relname     | parttype | parentid | boundaries
----------------+----------+----------+------------
 web_returns_p2 | r        |    97628 |
 p2016_0        | p        |    97628 | {20161231}
 p2016_1        | p        |    97628 | {20171231}
 p2016_2        | p        |    97628 | {20181231}
 p2016_3        | p        |    97628 | {20191231}
 p0             | p        |    97628 | {NULL}
(6 rows)
```

14.2.45 PG_PLTEMPLATE

PG_PLTEMPLATE records template information for procedural languages.

Table 14-47 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. If no name of the block handler exists, the value is null.
tmplvalidator	text	Name of the verification function. If no verification function is available, the value is null.

Name	Type	Description
tmpllibrary	text	Path of the shared library that implements languages
tmplacl	aclitem[]	Access permissions for template (not yet used)

14.2.46 PG_PROC

PG_PROC records information about functions or procedures.

Table 14-48 PG_PROC columns

Name	Type	Description
proname	name	Name of the function
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
prorows	real	Estimate number of result rows
provariadic	oid	Data type of parameter element
protransform	regproc	Simplified call method for this function
proisagg	boolean	Whether this function is an aggregate function
proiswindow	boolean	Whether this function is a window function
prosecdef	boolean	Whether this function is a security definer (such as a "setuid" function)
proleakproof	boolean	Whether this function has side effects. If no leakproof treatment is provided for parameters, the function throws errors.
proisstrict	boolean	The function returns null if any call parameter is null. In that case the function does not actually be called at all. Functions that are not "strict" must be prepared to process null inputs.
proretset	boolean	The function returns a set, that is, multiple values of the specified data type.

Name	Type	Description
provolatile	"char"	Whether the function's result depends only on its input parameters, or is affected by outside factors ● It is **i** for "immutable" functions, which always deliver the same result for the same inputs. ● It is **s** for "stable" functions, whose results (for fixed inputs) do not change within a scan. ● It is **v** for "volatile" functions, whose results may change at any time.
pronargs	smallint	Number of parameters
pronargdefaults	smallint	Number of parameters that have default values
prorettype	oid	OID of the returned parameter type
proargtypes	oidvector	Array with the data types of the function parameters. This array includes only input parameters (including **INOUT** parameters) and thus represents the call signature of the function.
proallargtypes	oid[]	Array with the data types of the 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, and **proargtypes** is subscripted from 0.
proargmodes	"char"[]	Array with the modes of the function parameters. ● **i** indicates **IN** parameters. ● **o** indicates **OUT** parameters. ● **b** indicates **INOUT** parameters. If all the parameters are **IN** parameters, this column is null. Note that subscripts of this array 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.

Name	Type	Description
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 calling 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.
prodefaultargpos	int2vector	Locations of the function default values. Not only the last few parameters have default values.
fencedmode	boolean	Execution mode of a function, indicating whether a 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. The default value is **fence**.
proshippable	boolean	Whether a function can be pushed down to DNs. The default value is **false**. ● Functions of the IMMUTABLE type can always be pushed down to the DNs. ● Functions of the STABLE or VOLATILE type can be pushed down to DNs only if their attribute is **SHIPPABLE**.
propackage	boolean	Indicates whether the function supports overloading, which is mainly used for the Oracle style function. The default value is **false**.

Examples

Query the OID of a specified function. For example, obtain the OID **1295** of the **justify_days** function.

```
SELECT oid FROM pg_proc WHERE proname ='justify_days';
 oid
------
 1295
(1 row)
```

Query whether a function is an aggregate function. For example, the **justify_days** function is a non-aggregate function.

```
SELECT proisagg FROM pg_proc WHERE proname ='justify_days';
 proisagg
----------
 f
(1 row)
```

Query the owner of a specified function. For example, the query returns that the owner of the **func_add_sql** function is user **u1**.

```
SELECT proowner FROM pg_proc WHERE proname='func_add_sql';
 proowner
----------
   542778
(1 row)

SELECT usename FROM pg_user WHERE usesysid = '542778';
 usename
---------
 u1
(1 row)
```

14.2.47 PG_RANGE

PG_RANGE records information about range types.

This is in addition to the types' entries in **PG_TYPE**.

Table 14-49 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, or 0 if none
rngsubopc	oid	**PG_OPCLASS**.oid	OID of the subtype's operator class used for range comparisons **rngsubopc** (plus **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**.
rngcanonical	regproc	**PG_PROC**.oid	OID of the function to convert a range value into canonical form, or 0 if none
rngsubdiff	regproc	**PG_PROC**.oid	OID of the function to return the difference between two element values as **double precision**, or 0 if none

14.2.48 PG_REDACTION_COLUMN

PG_REDACTION_COLUMN records the information about the masked columns.

Table 14-50 PG_REDACTION_COLUMN columns

Name	Type	Description
object_oid	oid	OID of the object to be masked
column_attrno	smallint	**attrno** of the masked column
function_type	integer	Masking type **NOTE** This column is reserved. It is used only for forward compatibility of masked column information in earlier versions. The value can be **0** (NONE) or **1** (FULL).
function_parameters	text	Parameters used when the masking type is **partial** (reserved).
regexp_pattern	text	Pattern string when the masking type is **regexp** (reserved).
regexp_replace_string	text	Replacement string when the masking type is **regexp** (reserved).
regexp_position	integer	Start and end replacement positions when the masking type is **regexp** (reserved).
regexp_occurrence	integer	Replacement times when the masking type is **regexp** (reserved).
regexp_match_parameter	text	Regular control parameter used when the masking type is **regexp** (reserved).
column_description	text	Description of the masked column
function_expr	pg_node_tree	Internal representation of the masking function.

Name	Type	Description
inherited	bool	Whether a masked column is inherited from another masked column.

14.2.49 PG_REDACTION_POLICY

PG_REDACTION_POLICY records information about the object to be redacted.

Table 14-51 PG_REDACTION_POLICY columns

Name	Type	Description
object_oid	oid	OID of the object to be redacted.
policy_name	name	Name of the redaction policy.
enable	boolean	Policy status (enabled or disabled) **NOTE** The value can be: ● **true**: enabled. ● **false**: disabled.
expression	pg_node_tree	Policy effective expression (for users)
policy_description	text	Description of a policy
inherited	bool	Whether a redaction policy is inherited from another redaction policy.

14.2.50 PG_RELFILENODE_SIZE

The **PG_RELFILENODE_SIZE** system catalog provides file-level space statistics. Each record in the catalog corresponds to a physical file on the disk and the size of the file.

Table 14-52 PG_RELFILENODE_SIZE columns

Name	Type	Description
databaseid	oid	OID of the database that the physical file belongs to If a system catalog is shared across databases, its value is **0**.

Name	Type	Description
tablespaceid	oid	Tablespace OID of the physical file
relfilenode	oid	Serial number of the physical file
backendid	integer	ID of the background thread that creates the physical file. Generally, the value is **-1**.
type	integer	Type of the physical file. ● The value **0** indicates a data file. ● The value **1** indicates an FSM file. ● The value **2** indicates a VM file. ● The value **3** indicates a BCM file. ● If the value greater than 4 indicates the total size of the data file and BCM file of the column in a column-store table.
filesize	bigint	Size of the physical file, in bytes.

14.2.51 PG_RLSPOLICY

PG_RLSPOLICY displays the information about row-level access control policies.

Table 14-53 PG_RLSPOLICY columns

Name	Type	Description
polname	name	Name of a row-level access control policy
polrelid	oid	Table OID of a row-level access control policy
polcmd	char	SQL operations affected by a row-level access control policy. The options are ***(ALL)**, **r(SELECT)**, **w(UPDATE)**, and **d(DELETE)**.
polpermissive	boolean	Type of a row-level access control policy **NOTE** Values of **polpermissive**: ● **true**: The row-level access control policy is a permissive policy. ● **false**: The row-level access control policy is a restrictive policy.
polroles	oid[]	OID of database user affected by a row-level access control policy
polqual	pg_node_tree	SQL condition expression of a row-level access control policy

14.2.52 PG_RESOURCE_POOL

PG_RESOURCE_POOL records information about database resource pools.

Table 14-54 PG_RESOURCE_POOL columns

Name	Type	Description
respool_name	name	Name of the resource pool
mem_percent	integer	Percentage of the memory configuration
cpu_affinity	bigint	Reserved column without an actual meaning
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 number of concurrent simple jobs allowed by the resource pool. **-1** and **0** indicate that there are no limitations.
memory_limit	name	Maximum memory of resource pool
parentid	oid	OID of the parent resource pool
io_limits	integer	Reserved column without an actual meaning
io_priority	text	Reserved column without an actual meaning
is_foreign	boolean	Indicates whether the resource pool can be used for users outside the logical cluster. If it is set to **true**, the resource pool controls the resources of common users who do not belong to the current resource pool.
short_acc	boolean	Whether to enable short query acceleration for a resource pool. This function is enabled by default. ● If short query acceleration is enabled, simple queries are controlled on the fast lane. ● If short query acceleration is disabled, and simple queries are controlled on the slow lane.

Name	Type	Description
except_rule	text	Exception rule associated with a resource pool. There can be multiple associated rules, which are separated by commas (,).

14.2.53 PG_REWRITE

PG_REWRITE records rewrite rules defined for tables and views.

Table 14-55 PG_REWRITE columns

Name	Type	Description
rulename	name	Name of the rule
ev_class	oid	Name of the table that uses the rule
ev_attr	smallint	Field to which the rule applies. Currently, the value is **0**, indicating the entire table.
ev_type	"char"	Event type for this rule: ● 1 = SELECT ● 2 = UPDATE ● 3 = INSERT ● 4 = DELETE
ev_enabled	"char"	Controls in which mode the rule fires ● **O**: The rule fires in "origin" and "local" modes. ● **D**: The rule is disabled. ● **R**: The rule fires in "replica" mode. ● **A**: The rule always fires.
is_instead	boolean	Its value is **true** if the rule is an **INSTEAD** rule.
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

14.2.54 PG_SECLABEL

PG_SECLABEL records security labels on database objects.

See also **PG_SHSECLABEL**, which performs a similar function for security labels of database objects that are shared across a database cluster.

Table 14-56 PG_SECLABEL columns

Name	Type	Reference	Description
objoid	oid	Any OID column	OID of the object this security label pertains to
classoid	oid	**PG_CLASS**.oid	OID of the system catalog that contains the object
objsubid	integer	-	For a security label on a table column, this is the column number.
provider	text	-	Label provider associated with this label
label	text	-	Security label applied to this object

14.2.55 PG_SHDEPEND

PG_SHDEPEND records the dependency relationships between database objects and shared objects, such as roles. This information allows GaussDB(DWS) to ensure that those objects are unreferenced before they are deleted.

See also **PG_DEPEND**, which performs a similar function for dependencies involving objects within a single database.

Unlike most system catalogs, **PG_SHDEPEND** is shared across all databases of a cluster: there is only one copy of **PG_SHDEPEND** per cluster, not one per database.

Table 14-57 PG_SHDEPEND columns

Name	Type	Reference	Description
dbid	oid	**PG_DATABASE**.oid	OID of the database the dependent object is in. The value is **0** for a shared object.
classid	oid	**PG_CLASS**.oid	OID of the system catalog the dependent object is in.
objid	oid	Any OID column	OID of the specific dependent object
objsubid	integer	-	For a table column, this is the column number (the **objid** and **classid** refer to the table itself). For all other object types, this column is **0**.

Name	Type	Reference	Description
refclassid	oid	PG_CLASS.oid	OID of the system catalog the referenced object is in (must be a shared catalog)
refobjid	oid	Any OID column	OID of the specific referenced object
deptype	"char"	-	Code segment defining the specific semantics of this dependency relationship. See the following text for details.
objfile	text	-	Path of the user-defined C 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 defined 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 ACL (access control list, i.e., privileges list) 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.

14.2.56 PG_SHDESCRIPTION

PG_SHDESCRIPTION records optional comments for shared database objects. Descriptions can be manipulated with the **COMMENT** command and viewed with gsql's **\d** commands.

See also PG_DESCRIPTION, which performs a similar function for descriptions involving objects within a single database.

Unlike most system catalogs, **PG_SHDESCRIPTION** is shared across all databases of a cluster. There is only one copy of **PG_SHDESCRIPTION** per cluster, not one per database.

Table 14-58 PG_SHDESCRIPTION columns

Name	Type	Reference	Description
objoid	oid	Any OID column	OID of the object this description pertains to
classoid	oid	**PG_CLASS**.oid	OID of the system catalog where the object resides
description	text	-	Arbitrary text that serves as the description of this object

14.2.57 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 performs a similar function for security labels involving objects within a single database.

Unlike most system catalogs, **PG_SHSECLABEL** is shared across all databases of a cluster. There is only one copy of **PG_SHSECLABEL** per cluster, not one per database.

Table 14-59 PG_SHSECLABEL columns

Name	Type	Reference	Description
objoid	oid	Any OID column	OID of the object this security label pertains to
classoid	oid	**PG_CLASS**.oid	OID of the system catalog where the object resides
provider	text	-	Label provider associated with this label
label	text	-	Security label applied to this object

14.2.58 PG_STATISTIC

PG_STATISTIC records statistics about tables and index columns in a database. It is accessible only to users with system administrator rights.

Table 14-60 PG_STATISTIC columns

Name	Type	Description
starelid	oid	Table or index which 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, not-null data values in the column for all DNs ● A value greater than zero is the actual number of distinct values. ● A value less than zero 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.) ● **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 kind 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 zero is the actual number of distinct values. ● A value less than zero 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.) ● **0** indicates that the number of distinct values is unknown.
staextinfo	text	Information about extension statistics (reserved)

14.2.59 PG_STATISTIC_EXT

PG_STATISTIC_EXT records extended statistics about tables in a database. The range of extended statistics to be collected is specified by users. Only system administrators can access this system catalog.

Table 14-61 PG_STATISTIC_EXT columns

Parameter	Type	Description
starelid	oid	Table or index which the described column belongs to
starelkind	"char"	Type of an object
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, not-null data values in the column for all DNs ● A value greater than zero is the actual number of distinct values. ● A value less than zero 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.) ● **0** indicates that the number of distinct values is unknown.

Parameter	Type	Description
stadndistinct	real	Number of unique non-null data values in the **dn1** column ● A value greater than zero is the actual number of distinct values. ● A value less than zero 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.) ● **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 kind 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.

14.2.60 PG_SYNONYM

PG_SYNONYM records the mapping between synonym object names and other database object names.

Table 14-62 PG_SYNONYM columns

Name	Type	Description
synname	name	Synonym name.

Name	Type	Description
synnamespace	oid	OID of the namespace where the synonym is located.
synowner	oid	Owner of a synonym, usually the OID of the user who created it.
synobjschema	name	Schema name specified by the associated object.
synobjname	name	Name of the associated object.

14.2.61 PG_TABLESPACE

PG_TABLESPACE records tablespace information.

Table 14-63 PG_TABLESPACE columns

Name	Type	Description
spcname	name	Name of the tablespace
spcowner	oid	Owner of the tablespace, usually the user who created it
spcacl	aclitem[]	Access permissions For details, see GRANT and REVOKE.
spcoptions	text[]	Specifies options of the tablespace.
spcmaxsize	text	Maximum size of the available disk space, in bytes

14.2.62 PG_TRIGGER

PG_TRIGGER records the trigger information.

Name	Type	Description
tgrelid	oid	OID of the table where the trigger is located.
tgname	name	Trigger name.
tgfoid	oid	Trigger OID.
tgtype	smallint	Trigger type

Name	Type	Description
tgenabled	"char"	**O**: The trigger fires in "origin" or "local" mode. **D**: The trigger is disabled. **R**: The trigger fires in "replica" mode. **A**: The trigger always fires.
tgisinternal	boolean	Internal trigger ID. If the value is true, it indicates an internal trigger.
tgconstrrelid	oid	The table referenced by the integrity constraint
tgconstrindid	oid	Index of the integrity constraint
tgconstraint	oid	OID of the constraint trigger in the **pg_constraint**
tgdeferrable	boolean	The constraint trigger is of the DEFERRABLE type.
tginitdeferred	boolean	whether the trigger is of the INITIALLY DEFERRED type
tgnargs	smallint	Input parameters number 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	Indicates the WHEN condition of the trigger. If the WHEN condition does not exist, the value is null.

14.2.63 PG_TS_CONFIG

PG_TS_CONFIG records 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 14-64 PG_TS_CONFIG columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute; must be explicitly selected)

Name	Type	Reference	Description
cfgname	name	-	Text search configuration name
cfgnamespace	oid	**PG_NAMESPACE**.oid	OID of the namespace where the configuration resides
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

14.2.64 PG_TS_CONFIG_MAP

PG_TS_CONFIG_MAP records 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 14-65 PG_TS_CONFIG_MAP columns

Name	Type	Reference	Description
mapcfg	oid	**PG_TS_CONFIG**.oid	OID of the **PG_TS_CONFIG** entry owning this map entry
maptokentype	integer	-	A token type emitted by the configuration's parser
mapseqno	integer	-	Order in which to consult this entry
mapdict	oid	**PG_TS_DICT**.oid	OID of the text search dictionary to consult

14.2.65 PG_TS_DICT

PG_TS_DICT records 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 14-66 PG_TS_DICT columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute; must be explicitly selected)
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 this dictionary
dictinitoption	text	-	Initialization option string for the template

14.2.66 PG_TS_PARSER

PG_TS_PARSER records entries defining text search parsers. A parser splits input text into lexemes and assigns a token type to each lexeme. Since a parser must be implemented by C functions, parsers can be created only by database administrators.

Table 14-67 PG_TS_PARSER columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute; must be explicitly selected)
prsname	name	-	Text search parser name
prsnamespace	oid	**PG_NAMESPACE**.oid	OID of the namespace that contains the parser
prsstart	regproc	**PG_PROC**.oid	OID of the parser's startup function
prstoken	regproc	**PG_PROC**.oid	OID of the parser's next-token function
prsend	regproc	**PG_PROC**.oid	OID of the parser's shutdown function
prsheadline	regproc	**PG_PROC**.oid	OID of the parser's headline function

Name	Type	Reference	Description
prslextype	regproc	**PG_PROC**.oid	OID of the parser's lextype function

14.2.67 PG_TS_TEMPLATE

PG_TS_TEMPLATE records entries defining text search templates. A template provides a framework for text search dictionaries. Since a template must be implemented by C functions, templates can be created only by database administrators.

Table 14-68 PG_TS_TEMPLATE columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute; must be explicitly selected)
tmplname	name	-	Text search template name
tmplnamespace	oid	**PG_NAMESPACE**.oid	OID of the namespace that contains the template
tmplinit	regproc	**PG_PROC**.oid	OID of the template's initialization function
tmpllexize	regproc	**PG_PROC**.oid	OID of the template's lexize function

14.2.68 PG_TYPE

PG_TYPE records the information about data types.

Table 14-69 PG_TYPE columns

Name	Type	Description
typname	name	Data type name
typnamespace	oid	OID of the namespace that contains this type
typowner	oid	Owner of this type

Name	Type	Description
typlen	smallint	Number of bytes in the internal representation of the type for a fixed-size type. But for a variable-length type, **typlen** is negative. ● **-1** indicates a "varlena" type (one that has a length word). ● **-2** indicates a null-terminated C string.
typbyval	boolean	Whether the value of this type is passed by parameter or reference of this column. **TYPBYVAL** is false if the type 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 the **TYPLEN** is passed by parameter of this column.
typtype	char	● **b** indicates a basic type. ● **c** indicates a composite type, for example, a table's row type. ● **e** indicates an enumeration type. ● **p** indicates a pseudo 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	Whether data is converted. It is **true** if conversion is performed when data meets the conversion rules specified by **TYPCATEGORY**.
typisdefined	boolean	The value is **true** if the type is defined. The value is **false** if this is a placeholder entry for a not-yet-defined type. When it is **false**, type name, namespace, and OID are the only dependable objects.
typdelim	"char"	Character that separates two values of this type when parsing array input. Note that the delimiter is associated with the array element data type, not the array data type.
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. The value is **0** for non-composite types.

Name	Type	Description
typelem	oid	If **typelem** is not 0 then it identifies another row in **pg_type**. The current type can be subscripted like an array yielding values of type **typelem**. The current type can then be subscripted like an array yielding values of type **typelem**. A "true" array type is variable length (**typlen** = -1), but some fixed-length (**typlen** > 0) types also have nonzero **typelem**, for example **name** and **point**. If a fixed-length type has a **typelem**, its internal representation must be some 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	Indicates that the corresponding type record is available in **pg_type** if the value is not **0**.
typinput	regproc	Input conversion function (text format)
typoutput	regproc	Output conversion function (text format)
typreceive	regproc	Input conversion function (binary format). If no input conversion function, the value is **0**.
typsend	regproc	output conversion function (binary format). If no output conversion function, the value is **0**.
typmodin	regproc	Type modifier input function. The value is **0** if the type does not support modifiers.
typmodout	regproc	Type modifier output function. The value is **0** if the type does not support modifiers.
typanalyze	regproc	Custom **ANALYZE** function. The value is **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 disk as well as most representations of the value inside PostgreSQL. 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**: **int** 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** indicates that values are always stored plain. ● **e**: Value 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).
typenotnull	boolean	Represents a **NOTNULL** constraint on a type. 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**). The value 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 (that is, **typbasetype** is an array type; the domain's **typelem** matches the base type's **typelem**). The value is **0** for types other than domains over array types.
typcollation	oid	Sequence rule for specified types. Sequencing is not supported if the value is 0.
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 human-readable version of the 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 permissions

14.2.69 PG_USER_MAPPING

PG_USER_MAPPING records the mappings from local users to remote.

It is accessible only to users with system administrator rights. You can use view **PG_USER_MAPPINGS** to query common users.

Table 14-70 PG_USER_MAPPING columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute; must be explicitly selected)
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 this mapping
umoptions	text[]	-	Option used for user mapping. It is a keyword=value string.

14.2.70 PG_USER_STATUS

PG_USER_STATUS records the states of users that access to the database. It is accessible only to users with system administrator rights.

Table 14-71 PG_USER_STATUS columns

Name	Type	Description
roloid	oid	ID of the role
failcount	integer	Specifies the number of failed attempts.
locktime	timestamp with time zone	Time at which the role is locked
rolstatus	smallint	Role state ● **0**: normal ● **1** indicates that the role is locked for some time because the failed login attempts exceed the threshold ● **2** indicates that the role is locked by the administrator.
permspace	bigint	Size of the permanent table storage space used by a role in the current instance.
tempspace	bigint	Size of the temporary table storage space used by a role in the current instance.

14.2.71 PG_WORKLOAD_ACTION

PG_WORKLOAD_ACTION records information about **query_band**.

Table 14-72 PG_WORKLOAD_ACTION columns

Name	Type	Description
qband	name	**query_band** key-value pairs
class	name	Class of the object associated with **query_band**
object	name	Object associated with **query_band**
action	name	Action of the object associated with **query_band**

14.2.72 PGXC_CLASS

PGXC_CLASS records the replicated or distributed information for each table.

Table 14-73 PGXC_CLASS columns

Name	Type	Description
pcrelid	oid	Table OID
pclocatortype	"char"	Locator type ● **H**: hash ● **M**: Modulo ● **N**: Round Robin ● **R**: Replicate
pchashalgorithm	smallint	Distributed tuple using the hash algorithm
pchashbuckets	smallint	Value of a harsh container
pgroup	name	Name of the node group
redistributed	"char"	The table has been redistributed.
redis_order	integer	Redistribution sequence
pcattnum	int2vector	Column number used as a distribution key
nodeoids	oidvector_extend	List of distributed table node OIDs
options	text	Extension status information. This is a reserved column in the system.

14.2.73 PGXC_GROUP

PGXC_GROUP records information about node groups.

Table 14-74 PGXC_GROUP columns

Name	Type	Description
group_name	name	Node Group name.
in_redistribution	"char"	Whether redistribution is required ● **n** indicates that the Node Group is not redistributed. ● **y** indicates the source Node Group in redistribution. ● **t** indicates the destination Node Group in redistribution.

Name	Type	Description
group_members	oidvector_extend	Node OID list of the Node Group
group_buckets	text	Distributed data bucket group
is_installation	boolean	Whether to install a sub-cluster
group_acl	aclitem[]	Access permissions
group_kind	"char"	Node Group type ● **i** indicates an installation Node Group. ● **n** indicates a Node Group in a common, non-logical cluster. ● **v** indicates a Node Group in a logical cluster. ● **e** indicates an elastic cluster.

14.2.74 PGXC_NODE

PGXC_NODE records information about cluster nodes.

Table 14-75 PGXC_NODE columns

Name	Type	Description
node_name	name	Node name
node_type	"char"	Node type **C**: CN **D**: DN
node_port	integer	Port ID 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 switchover occurs between the primary and the standby server on the current node

Name	Type	Description
nodeis_primary	boolean	Whether the current node is preferred to execute non-query operations in the **replication** table
nodeis_preferred	boolean	Whether the current node is preferred to execute queries in the **replication** table
node_id	integer	Node identifier
sctp_port	integer	Specifies the port used by the TCP proxy communication library or SCTP communication library of the primary node to listen to the data channel.
control_port	integer	Specifies the port used by the TCP proxy communication library or SCTP communication library of the primary node to listen to the control channel.
sctp_port1	integer	Specifies the port used by the TCP proxy communication library or SCTP communication library of the standby node to listen to the data channel.
control_port1	integer	Specifies the port used by the TCP proxy communication library or SCTP communication library of the standby node to listen to the control channel.
nodeis_central	boolean	Indicates that the current node is the central node.

Example

Query the number of DNs on a node:

```
SELECT count(node_name),node_host FROM pgxc_node WHERE node_type='D' GROUP BY 2;
 count |   node_host
-------+---------------
     1 | 192.**.**.10
     1 | 192.**.**.11
     1 | 192.**.**.12
(3 rows)
```

Query the CN and DN information of the cluster:

```
SELECT * FROM pgxc_node;
  node_name   | node_type | node_port |   node_host    | node_port1 |   node_host1   | hostis_primary |
nodeis_primary | nodeis_preferred
 |   node_id   | sctp_port | control_port | sctp_port1 | control_port1 | nodeis_central
--------------+-----------+-----------+----------------+------------+----------------+----------------+----------------
+-----------------
-+-------------+-----------+--------------+------------+---------------+----------------
 dn_6001_6002 | D         |     40000 | 192.**.***.**1 |      45000 | 192.**.**.**2  | t              | f              | f
 |  1644780306 |     40002 |        40003 |      45002 |         45003 | f
 dn_6003_6004 | D         |     40000 | 192.**.**.**2  |      45000 | 192.**.**.**3  | t              | f              | f
 |  -966646068 |     40002 |        40003 |      45002 |         45003 | f
```

```
dn_6005_6006 | D         |     40000 | 192.**.**.**3  |      45000 | 192.**.***.**1 | t              | f              | f
|   868850011 |     40002 |        40003 |      45002 |          45003 | f
cn_5001       | C         |      8000 | 192.**.***.**1 |       8000 | 192.**.***.**1 | t              | f              | f
|  1120683504 |      8002 |         8003 |          0 |              0 | f
cn_5002       | C         |      8000 | 192.**.**.**2  |       8000 | 192.**.**.**2  | t              | f              | f
| -1736975100 |      8002 |         8003 |          0 |              0 | f
cn_5003       | C         |      8000 | localhost      |       8000 | localhost      | t              | f              | f
|  -125853378 |      8002 |         8003 |          0 |              0 | t
(6 rows)
```

14.2.75 PLAN_TABLE_DATA

PLAN_TABLE_DATA stores the plan information collected by **EXPLAIN PLAN**. Different from the **PLAN_TABLE** view, the system catalog **PLAN_TABLE_DATA** stores the plan information collected by all sessions and users.

Table 14-76 PLAN_TABLE 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
plan_id	bigint	ID of a plan to be queried
id	int	Node ID in a plan
operation	varchar2(30)	Operation description
options	varchar2(255)	Operation parameters
object_name	name	Name of an operated object. It is defined by users.
object_type	varchar2(30)	Object type
object_owner	name	User-defined schema to which an object belongs
projection	varchar2(4000)	Returned column information

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 only their own data in the **PLAN_TABLE** view.
- Data of inactive (exited) sessions is cleaned from **PLAN_TABLE_DATA** by **gs_clean** after being stored in this system catalog for a certain period of time (5 minutes by default). You can also manually run **gs_clean -C** to delete inactive session data from the table..
- 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.

14.2.76 SNAPSHOT

SNAPSHOT records the start and end time of each performance view snapshot creation. After **enable_wdr_snapshot** is set to **on**, this catalog is created and maintained by the background snapshot thread. It is accessible only to users with system administrator rights.

Table 14-77 dbms_om.snapshot columns

Name	Type	Description
snapshot_id	name	Snapshot ID. This column is the primary key and distribution key.
start_ts	timestamp with time zone	Snapshot start time
end_ts	timestamp with time zone	Snapshot end time

NOTICE

- This system catalog's schema is **dbms_om**.
- Do not modify or delete this catalog externally. Otherwise, functions related to view snapshots may not work properly.

14.2.77 TABLES_SNAP_TIMESTAMP

TABLES_SNAP_TIMESTAMP records the start and end time of the snapshots created for each performance view. After **enable_wdr_snapshot** is set to **on**, this catalog is created and maintained by the background snapshot thread. It is accessible only to users with system administrator rights.

Table 14-78 dbms_om.tables_snap_timestamp columns

Name	Type	Description
snapshot_id	name	Snapshot ID. This column is the primary key and distribution key.
db_name	text	Name of the database to which the view belongs
tablename	text	View name
start_ts	timestamp with time zone	Snapshot start time
end_ts	timestamp with time zone	Snapshot end time

NOTICE

- This system catalog's schema is **dbms_om**.
- Do not modify or delete this catalog externally. Otherwise, functions related to view snapshots may not work properly.

14.2.78 System Catalogs for Performance View Snapshot

After **enable_wdr_snapshot** is set to **on**, the background snapshot thread creates and maintains a system catalog named in the format of **SNAP_***View name* to record the snapshot result of each performance view. The following system catalogs are accessible only to users with system administrator rights:

- SNAP_**PGXC_OS_RUN_INFO**
- SNAP_**PGXC_WAIT_EVENTS**
- SNAP_**PGXC_INSTR_UNIQUE_SQL**
- SNAP_**PGXC_STAT_BAD_BLOCK**
- SNAP_**PGXC_STAT_BGWRITER**
- SNAP_**PGXC_STAT_REPLICATION**
- SNAP_**PGXC_REPLICATION_SLOTS**
- SNAP_**PGXC_SETTINGS**
- SNAP_**PGXC_INSTANCE_TIME**
- SNAP_**GLOBAL_WORKLOAD_TRANSACTION**
- SNAP_**PGXC_WORKLOAD_SQL_COUNT**
- SNAP_**PGXC_STAT_DATABASE**
- SNAP_**GLOBAL_STAT_DATABASE**
- SNAP_**PGXC_REDO_STAT**
- SNAP_**GLOBAL_REDO_STAT**
- SNAP_**PGXC_REL_IOSTAT**

- SNAP_GLOBAL_REL_IOSTAT
- SNAP_PGXC_TOTAL_MEMORY_DETAIL
- SNAP_PGXC_NODE_STAT_RESET_TIME
- SNAP_PGXC_SQL_COUNT
- SNAP_GLOBAL_TABLE_STAT
- SNAP_GLOBAL_TABLE_CHANGE_STAT
- SNAP_GLOBAL_COLUMN_TABLE_IO_STAT
- SNAP_GLOBAL_ROW_TABLE_IO_STAT

Except the new **snapshot_id** column (of the bigint type), the definitions of the other columns in these system catalogs are the same as those of the corresponding views, and the distribution key of each system catalog is **snapshot_id**.

For example, **SNAP_PGXC_OS_RUN_INFO** is used to record snapshots of the **PGXC_OS_RUN_INFO** view. The **snapshot_id** column is new, and other columns are the same as those of the **PGXC_OS_RUN_INFO** view.

NOTICE

- The schema of all above system catalogs is **dbms_om**.
- Do not modify or delete these catalogs externally. Otherwise, functions related to view snapshots may not work properly.

14.3 System Views

14.3.1 ALL_ALL_TABLES

ALL_ALL_TABLES displays the tables or views accessible to the current user.

Table 14-79 ALL_ALL_TABLES columns

Name	Type	Description
owner	name	Owner of a table/view
table_name	name	Name of the table or the view
tablespace_name	name	Tablespace where the table or view is located

14.3.2 ALL_CONSTRAINTS

ALL_CONSTRAINTS displays information about constraints accessible to the current user.

Table 14-80 ALL_CONSTRAINTS columns

Name	Type	Description
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 constraint-related table
index_owner	character varying(64)	Owner of constraint-related index (only for the unique constraint and primary key constraint)
index_name	character varying(64)	Name of constraint-related index (only for the unique constraint and primary key constraint)

14.3.3 ALL_CONS_COLUMNS

ALL_CONS_COLUMNS displays information about constraint columns accessible to the current user.

Table 14-81 ALL_CONS_COLUMNS columns

Name	Type	Description
table_name	character varying(64)	Name of constraint-related table
column_name	character varying(64)	Name of constraint-related column
constraint_name	character varying(64)	Constraint name
position	smallint	Position of the column in the table

14.3.4 ALL_COL_COMMENTS

ALL_COL_COMMENTS displays column comments of tables and views that the current user can access.

Table 14-82 ALL_COL_COMMENTS columns

Name	Type	Description
column_name	character varying(64)	Column name
table_name	character varying(64)	Table/View name
owner	character varying(64)	Owner of a table/view
comments	text	Comments

14.3.5 ALL_DEPENDENCIES

ALL_DEPENDENCIES displays dependencies between functions and advanced packages accessible to the current user.

NOTICE

Currently in GaussDB(DWS), this table is empty without any record due to information constraints.

Table 14-83 ALL_DEPENDENCIES columns

Name	Type	Description
owner	character varying(30)	Owner of the object
name	character varying(30)	Object name
type	character varying(17)	Type of the object
referenced_owner	character varying(30)	Owner of the referenced object
referenced_name	character varying(64)	Name of the referenced object
referenced_type	character varying(17)	Type of the referenced object
referenced_link_name	character varying(128)	Name of the link to the referenced object
schemaid	numeric	ID of the current schema
dependency_type	character varying(4)	Dependency type (REF or HARD)

14.3.6 ALL_IND_COLUMNS

ALL_IND_COLUMNS displays all index columns accessible to the current user.

Table 14-84 ALL_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 column in the index

14.3.7 ALL_IND_EXPRESSIONS

ALL_IND_EXPRESSIONS displays information about the expression indexes accessible to the current user.

Table 14-85 ALL_IND_EXPRESSIONS 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_expression	text	Function-based index expression of a specified column
column_position	smallint	Position of a column in the index

14.3.8 ALL_INDEXES

ALL_INDEXES displays information about indexes accessible to the current user.

Table 14-86 ALL_INDEXES columns

Name	Type	Description
owner	character varying(64)	Index owner
index_name	character varying(64)	Index name

Name	Type	Description
table_name	character varying(64)	Name of the table corresponding to the index.
uniqueness	text	Whether the index is a unique index
generated	character varying(1)	Whether the index name is generated by the system
partitioned	character(3)	Whether the index has the property of the partition table

14.3.9 ALL_OBJECTS

ALL_OBJECTS displays all database objects accessible to the current user.

Table 14-87 ALL_OBJECTS columns

Name	Type	Description
owner	name	Object owner
object_name	name	Object name
object_id	oid	OID of the object
object_type	name	Type of the object
namespace	oid	Namespace containing the object
created	timestamp with time zone	Object creation time
last_ddl_time	timestamp with time zone	The last time when an object was modified.

NOTICE

For details about the value ranges of **last_ddl_time** and **last_ddl_time**, see **PG_OBJECT**.

14.3.10 ALL_PROCEDURES

ALL_PROCEDURES displays information about all stored procedures or functions accessible to the current user.

Table 14-88 ALL_PROCEDURES columns

Name	Type	Description
owner	name	Object owner
object_name	name	Object name

14.3.11 ALL_SEQUENCES

ALL_SEQUENCES displays all sequences accessible to the current user.

Table 14-89 ALL_SEQUENCES columns

Name	Type	Description
sequence_owner	name	Owner of the sequence
sequence_name	name	Name of the sequence
min_value	bigint	Minimum value of the sequence
max_value	bigint	Maximum value of the sequence
increment_by	bigint	Value by which the 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.

14.3.12 ALL_SOURCE

ALL_SOURCE displays information about stored procedures or functions accessible to the current user, and provides the columns defined by the stored procedures and functions.

Table 14-90 ALL_SOURCE columns

Name	Type	Description
owner	name	Object owner
name	name	Object name
type	name	Object type
text	text	Object definition

14.3.13 ALL_SYNONYMS

ALL_SYNONYMS displays all synonyms accessible to the current user.

Table 14-91 ALL_SYNONYMS columns

Name	Type	Description
owner	text	Owner of a synonym.
schema_name	text	Name of the schema to which the synonym belongs
synonym_name	text	Synonym name
table_owner	text	Owner of the associated object
table_schema_name	text	Schema name of the associated object
table_name	text	Name of the associated object

14.3.14 ALL_TAB_COLUMNS

ALL_TAB_COLUMNS displays description of columns of the tables and views that the current user can access.

Table 14-92 ALL_TAB_COLUMNS columns

Name	Type	Description
owner	character varying(64)	Owner of a table/view
table_name	character varying(64)	Table/View name
column_name	character varying(64)	Column name
data_type	character varying(128)	Data type of a column
column_id	integer	Column ID generated when an object is created or a column is added
data_length	integer	Length of the column, in bytes
avg_col_len	numeric	Average length of a column, in bytes
nullable	bpchar	Whether the column can be empty. For the primary key constraint and non-null constraint, the value is n.

Name	Type	Description
data_precision	integer	Precision of the data type. This parameter is valid for the numeric data type and **NULL** for other types.
data_scale	integer	Number of decimal places. This parameter is valid for the numeric data type and **0** for other data types.
char_length	numeric	Length of a column, in characters. This parameter is valid only for the varchar, nvarchar2, bpchar, and char types.
schema	character varying(64)	Namespace that contains the table or view.
kind	text	Type of the current record. If the column belongs to a table, the value of this column is **table**. If the column belongs to a view, the value of this column is **view**.

14.3.15 ALL_TAB_COMMENTS

ALL_TAB_COMMENTS displays comments about all tables and views accessible to the current user.

Table 14-93 ALL_TAB_COMMENTS columns

Name	Type	Description
owner	character varying(64)	Owner of a table/view
table_name	character varying(64)	Name of the table or the view
comments	text	Comments

14.3.16 ALL_TABLES

ALL_TABLES displays all the tables accessible to the current user.

Table 14-94 ALL_TABLES columns

Name	Type	Description
owner	character varying(64)	Table owner
table_name	character varying(64)	Table name
tablespace_name	character varying(64)	Name of the tablespace that contains the table

Name	Type	Description
status	character varying(8)	Whether the current record is valid
temporary	character(1)	Whether the table is a temporary table ● **Y** indicates that it is a temporary table. ● **N** indicates that it is not a temporary table.
dropped	character varying	Whether the current record is deleted ● **YES** indicates that it is deleted. ● **NO** indicates that it is not deleted.
num_rows	numeric	Estimated number of rows in the table

14.3.17 ALL_USERS

ALL_USERS displays all users of the database visible to the current user, however, it does not describe the users.

Table 14-95 ALL_USERS columns

Name	Type	Description
username	name	Name of the user
user_id	oid	OID of the user

14.3.18 ALL_VIEWS

ALL_VIEWS displays the description about all views accessible to the current user.

Table 14-96 ALL_VIEWS columns

Name	Type	Description
owner	name	Owner of the view
view_name	name	Name of the view
text_length	integer	Text length of the view

Name	Type	Description
text	text	Text in the view

14.3.19 DBA_DATA_FILES

DBA_DATA_FILES displays the description of database files. It is accessible only to users with system administrator rights.

Table 14-97 DBA_DATA_FILES columns

Name	Type	Description
tablespace_name	name	Name of the tablespace to which the file belongs
bytes	double precision	Length of the file in bytes

14.3.20 DBA_USERS

DBA_USERS displays all user names in the database. It is accessible only to users with system administrator rights.

Table 14-98 DBA_USERS columns

Name	Type	Description
username	character varying(64)	Name of the user

14.3.21 DBA_COL_COMMENTS

DBA_COL_COMMENTS displays column comments in the tables and views of a database. Only users with system administrator permissions can access this view.

Name	Type	Description
column_name	character varying(64)	Column name
table_name	character varying(64)	Table/View name
owner	character varying(64)	Owner of a table/view
comments	text	Comments

14.3.22 DBA_CONSTRAINTS

DBA_CONSTRAINTS displays information about table constraints in database. It is accessible only to users with system administrator rights.

Name	Type	Description
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 constraint-related table
index_owner	character varying(64)	Owner of constraint-related index (only for the unique constraint and primary key constraint)
index_name	character varying(64)	Name of constraint-related index (only for the unique constraint and primary key constraint)

14.3.23 DBA_CONS_COLUMNS

DBA_CONS_COLUMNS displays information about constraint columns in database tables. It is accessible only to users with system administrator rights.

Name	Type	Description
table_name	character varying(64)	Name of constraint-related table
column_name	character varying(64)	Name of constraint-related column
constraint_name	character varying(64)	Constraint name
position	smallint	Position of the column in the table

14.3.24 DBA_IND_COLUMNS

DBA_IND_COLUMNS displays column information about all indexes in the database. It is accessible only to users with system administrator rights.

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 column in the index

14.3.25 DBA_IND_EXPRESSIONS

DBA_IND_EXPRESSIONS displays the information about expression indexes in the database. It is accessible only to users with system administrator rights.

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_expression	text	The function-based index expression of a specified column
column_position	smallint	Position of column in the index

14.3.26 DBA_IND_PARTITIONS

DBA_IND_PARTITIONS displays information about all index partitions in the database. Each index partition of a partitioned table in the database, if present, has a row of records in **DBA_IND_PARTITIONS**. This view is accessible only to users with system administrator rights.

Name	Type	Description
index_owner	character varying(64)	Name of the owner of the partitioned index to which the index partition belongs
schema	character varying(64)	Schema of the partitioned index to which the index partition belongs

Name	Type	Description
index_name	character varying(64)	Index name of the partitioned table to which the index partition belongs
partition_name	character varying(64)	Name of the index partition
index_partition_usable	boolean	Whether the index partition is available
high_value	text	Boundary of the table partition corresponding to the index partition. For a range partition, the boundary is the upper boundary. For a list partition, the boundary is the boundary value set. Reserved field for forward compatibility. The parameter **pretty_high_value** is added in version 8.1.3 to record the information.
pretty_high_value	text	Boundary of the table partition corresponding to the index partition. For a range partition, the boundary is the upper boundary. For a list partition, the boundary is the boundary value set. The query result is the instant decompilation output of the partition boundary expression. The output of this column is more detailed than that of **high_value**. The output information can be collation and column data type.
def_tablespace_name	name	Tablespace name of the index partition

14.3.27 DBA_INDEXES

DBA_INDEXES displays all indexes in the database. This view is accessible only to users with system administrator rights.

Name	Type	Description
owner	character varying(64)	Owner of the index
index_name	character varying(64)	Index name
table_name	character varying(64)	Name of the table corresponding to the index
uniqueness	text	Whether the index is a unique index

Name	Type	Description
generated	character varying(1)	Whether the index name is generated by the system
partitioned	character(3)	Whether the index has the property of the partition table

14.3.28 DBA_OBJECTS

DBA_OBJECTS displays all database objects in the database. This view is accessible only to users with system administrator rights.

Name	Type	Description
owner	name	Owner of the object
object_name	name	Object name
object_id	oid	OID of the object
object_type	name	Type of the object
namespace	oid	Namespace containing the object
created	timestamp with time zone	Object creation time
last_ddl_time	timestamp with time zone	The last time when an object was modified.

NOTICE

For details about the value ranges of **last_ddl_time** and **last_ddl_time**, see **PG_OBJECT**.

14.3.29 DBA_PART_INDEXES

DBA_PART_INDEXES displays information about all partitioned table indexes in the database. It is accessible only to users with system administrator rights.

Name	Type	Description
index_owner	character varying(64)	Name of the owner of the partitioned table index

Name	Type	Description
schema	character varying(64)	Schema of the partitioned table index
index_name	character varying(64)	Name of the partitioned table index
table_name	character varying(64)	Name of the partitioned table to which the partitioned table index belongs
partitioning_type	text	Partition policy of the partitioned table **NOTE** Currently, only range partitioning and list partitioning are supported.
partition_count	bigint	Number of index partitions of the partitioned table index
def_tablespace_name	name	Tablespace name of the partitioned table index
partitioning_key_count	integer	Number of partition keys of the partitioned table

14.3.30 DBA_PART_TABLES

DBA_PART_TABLES displays information about all partitioned tables in the database. It is accessible only to users with system administrator rights.

Name	Type	Description
table_owner	character varying(64)	Name of the owner of the partitioned table
schema	character varying(64)	Schema of the partitioned table
table_name	character varying(64)	Name of the partitioned table
partitioning_type	text	Partition policy of the partitioned table **NOTE** Currently, only range partitioning and list partitioning are supported.
partition_count	bigint	Number of partitions of the partitioned table

Name	Type	Description
def_tablespace_name	name	Tablespace name of the partitioned table
partitioning_key_count	integer	Number of partition keys of the partitioned table

14.3.31 DBA_PROCEDURES

DBA_PROCEDURES displays information about all stored procedures and functions in the database. This view is accessible only to users with system administrator rights.

Name	Type	Description
owner	character varying(64)	Owner of the stored procedure or the function
object_name	character varying(64)	Name of the stored procedure or the function
argument_number	smallint	Number of the input parameters in the stored procedure

14.3.32 DBA_SEQUENCES

DBA_SEQUENCES displays information about all sequences in the database. This view is accessible only to users with system administrator rights.

Name	Type	Description
sequence_owner	character varying(64)	Owner of the sequence
sequence_name	character varying(64)	Name of the sequence

14.3.33 DBA_SOURCE

DBA_SOURCE displays all stored procedures or functions in the database, and it provides the columns defined by the stored procedures or functions. It is accessible only to users with system administrator rights.

Name	Type	Description
owner	character varying(64)	Owner of the stored procedure or the function

Name	Type	Description
name	character varying(64)	Name of the stored procedure or the function
text	text	Definition of the stored procedure or the function

14.3.34 DBA_SYNONYMS

DBA_SYNONYMS displays all synonyms in the database. It is accessible only to users with system administrator rights.

Table 14-99 DBA_SYNONYMS columns

Name	Type	Description
owner	text	Owner of a synonym
schema_name	text	Name of the schema to which the synonym belongs
synonym_name	text	Synonym name
table_owner	text	Owner of the associated object
table_schema_name	text	Schema name of the associated object
table_name	text	Name of the associated object

14.3.35 DBA_TAB_COLUMNS

DBA_TAB_COLUMNS stores the columns of tables and views. Each column of a table in the database has a row in **DBA_TAB_COLUMNS**. Only users with system administrator permissions can access this view.

Name	Type	Description
owner	character varying(64)	Owner of a table/view
table_name	character varying(64)	Table/View name
column_name	character varying(64)	Column name
data_type	character varying(128)	Data type of the column

Name	Type	Description
column_id	integer	Sequence number of the column when a table/view is created
data_length	integer	Length of the column, in bytes
comments	text	Comments
avg_col_len	numeric	Average length of a column, in bytes
nullable	bpchar	Whether the column can be empty. For the primary key constraint and non-null constraint, the value is **n**.
data_precision	integer	Precision of the 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 **0** for other data types.
char_length	numeric	Length of a column, in characters. This parameter is valid only for the varchar, nvarchar2, bpchar, and char types.
schema	character varying(64)	Namespace that contains the table or view.
kind	text	Type of the current record. If the column belongs to a table, the value of this column is **table**. If the column belongs to a view, the value of this column is **view**.

14.3.36 DBA_TAB_COMMENTS

DBA_TAB_COMMENTS displays comments about all tables and views in the database. It is accessible only to users with system administrator rights.

Name	Type	Description
owner	character varying(64)	Owner of a table/view
table_name	character varying(64)	Name of the table or the view
comments	text	Comments

14.3.37 DBA_TAB_PARTITIONS

DBA_TAB_PARTITIONS displays information about all partitions in the database.

Name	Type	Description
table_owner	character varying(64)	Owner of the table that contains the partition
schema	character varying(64)	Schema of the partitioned table
table_name	character varying(64)	Table name
partition_name	character varying(64)	Name of the partition
high_value	text	Upper boundary of a range partition or boundary value set of a list partition Reserved field for forward compatibility. The parameter **pretty_high_value** is added in version 8.1.3 to record the information.
pretty_high_value	text	Upper boundary of a range partition or boundary value set of a list partition The query result is the instant decompilation output of the partition boundary expression. The output of this column is more detailed than that of **high_value**. The output information can be collation and column data type.
tablespace_name	name	Name of the tablespace that contains the partition

Example

View the partition information of a partitioned table:

```
CREATE TABLE web_returns_p1
(
    wr_returned_date_sk       integer,
    wr_returned_time_sk       integer,
    wr_item_sk                integer NOT NULL,
    wr_refunded_customer_sk   integer
)
WITH (orientation = column)
DISTRIBUTE BY HASH (wr_item_sk)
PARTITION BY RANGE (wr_returned_date_sk)
(
    PARTITION p2016 VALUES LESS THAN(20161231),
    PARTITION p2017 VALUES LESS THAN(20171231),
    PARTITION p2018 VALUES LESS THAN(20181231),
    PARTITION p2019 VALUES LESS THAN(20191231),
    PARTITION p2020 VALUES LESS THAN(maxvalue)
);
```

```
SELECT * FROM dba_tab_partitions WHERE table_name='web_returns_p1';
 table_owner | schema |   table_name   | partition_name | high_value | pretty_high_value |  tablespace_name
-------------+--------+----------------+----------------+------------+-------------------+--------------------
 dbadmin     | public | web_returns_p1 | p2016          | 20161231   | 20161231          | DEFAULT TABLESPACE
 dbadmin     | public | web_returns_p1 | p2017          | 20171231   | 20171231          | DEFAULT TABLESPACE
 dbadmin     | public | web_returns_p1 | p2018          | 20181231   | 20181231          | DEFAULT TABLESPACE
 dbadmin     | public | web_returns_p1 | p2019          | 20191231   | 20191231          | DEFAULT TABLESPACE
 dbadmin     | public | web_returns_p1 | p2020          | MAXVALUE   | MAXVALUE          | DEFAULT
TABLESPACE
(5 rows)
```

14.3.38 DBA_TABLES

DBA_TABLES displays all tables in the database. This view is accessible only to users with system administrator rights.

Name	Type	Description
owner	character varying(64)	Table owner
table_name	character varying(64)	Table name
tablespace_name	character varying(64)	Name of the tablespace that contains the table
status	character varying(8)	Whether the current record is valid
temporary	character(1)	Whether the table is a temporary table ● **Y** indicates that it is a temporary table. ● **N** indicates that it is not a temporary table.
dropped	character varying	Whether the current record is deleted ● **YES** indicates that it is deleted. ● **NO** indicates that it is not deleted.
num_rows	numeric	Estimated number of rows in the table

14.3.39 DBA_TABLESPACES

DBA_TABLESPACES displays information about available tablespaces. It is accessible only to users with system administrator rights.

Table 14-100 DBA_TABLESPACES columns

Name	Type	Description
tablespace_name	character varying(64)	Name of the tablespace

14.3.40 DBA_TRIGGERS

DBA_TRIGGERS displays information about triggers in the database. This view is accessible only to users with system administrator rights.

Name	Type	Description
trigger_name	character varying(64)	Trigger name
table_name	character varying(64)	Name of the table that defines the trigger
table_owner	character varying(64)	Owner of the table that defines the trigger

14.3.41 DBA_VIEWS

DBA_VIEWS displays views in the database. This view is accessible only to users with system administrator rights.

Name	Type	Description
owner	character varying(64)	Owner of the view
view_name	character varying(64)	View name

14.3.42 DUAL

DUAL 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. It is accessible to all users.

Table 14-101 DUAL columns

Name	Type	Description
dummy	text	Expression calculation result

14.3.43 GLOBAL_COLUMN_TABLE_IO_STAT

GLOBAL_COLUMN_TABLE_IO_STAT provides I/O statistics of all column-store tables in the current database. The names, types, and sequence of the columns are the same as those in the **GS_COLUMN_TABLE_IO_STAT** view. The value of each statistical column is the sum of the values of the corresponding columns of all nodes.

Table 14-102 GLOBAL_COLUMN_TABLE_IO_STAT columns

Name	Type	Description
schemaname	name	Table namespace
relname	name	Table name
heap_read	bigint	Number of blocks logically read in the heap
heap_hit	bigint	Number of block hits in the heap
idx_read	bigint	Number of blocks logically read in the index
idx_hit	bigint	Number of block hits in the index
cu_read	bigint	Number of logical reads in the Compression Unit
cu_hit	bigint	Number of hits in the Compression Unit
cidx_read	bigint	Number of indexes logically read in the Compression Unit
cidx_hit	bigint	Number of index hits in the Compression Unit

14.3.44 GLOBAL_REDO_STAT

GLOBAL_REDO_STAT displays the total statistics of XLOG redo operations on all nodes in a cluster. Except the **avgiotim** column (indicating the average redo write time of all nodes), the names of the other columns in this view are the same as those in the **PV_REDO_STAT** view. The respective meanings of the other columns are the sum of the values of the same columns in the **PV_REDO_STAT** view on each node.

Table 14-103 GLOBAL_REDO_STAT columns

Name	Type	Description
phywrts	bigint	Total number of physical writes on all nodes
phyblkwrt	bigint	Total number of physical write blocks on all nodes
writetim	bigint	Total physical write time of all nodes

Name	Type	Description
avgiotim	bigint	Average redo write time of all nodes
lstiotim	bigint	Sum of the last write time of all nodes
miniotim	bigint	Sum of the minimum write time of all nodes
maxiowtm	bigint	Sum of the maximum write time of all nodes

NOTE

This view is accessible only to users with system administrator rights.

14.3.45 GLOBAL_REL_IOSTAT

GLOBAL_REL_IOSTAT displays the total disk I/O statistics of all nodes in a cluster. The name of each column in this view is the same as that in the **GS_REL_IOSTAT** view, but the column meaning is the sum of the value of the same column in the **GS_REL_IOSTAT** view on each node.

Table 14-104 GLOBAL_REL_IOSTAT columns

Name	Type	Description
phyrds	bigint	Total number of disk read times of all nodes
phywrts	bigint	Total number of disk write times of all nodes
phyblkrd	bigint	Total number of disk pages read by all nodes
phyblkwrt	bigint	Total number of disk pages written by all nodes

NOTE

This view is accessible only to users with system administrator rights.

14.3.46 GLOBAL_ROW_TABLE_IO_STAT

GLOBAL_ROW_TABLE_IO_STAT provides I/O statistics of all row-store tables in the current database. The names, types, and sequences of the columns in the view are the same as those in the **GS_ROW_TABLE_IO_STAT** view. For details about the columns, see **Table 14-105**. The value of each statistical column is the sum of the values of the corresponding columns of all nodes.

Table 14-105 GS_ROW_TABLE_IO_STAT columns

Name	Type	Description
schemaname	name	Table namespace
relname	name	Table name
heap_read	bigint	Number of blocks logically read in the heap
heap_hit	bigint	Number of block hits in the heap
idx_read	bigint	Number of blocks logically read in the index
idx_hit	bigint	Number of block hits in the index
toast_read	bigint	Number of blocks logically read in the **TOAST** table
toast_hit	bigint	Number of block hits in the **TOAST** table
tidx_read	bigint	Number of indexes logically read in the **TOAST** table
tidx_hit	bigint	Number of index hits in the **TOAST** table

14.3.47 GLOBAL_STAT_DATABASE

GLOBAL_STAT_DATABASE displays the status and statistics of databases on all nodes in a cluster.

- When you query the **GLOBAL_STAT_DATABASE** view on a CN, the respective values of all columns returned, except **stats_reset** (indicating the status reset time on the current CN), are the sum of values on related nodes in the cluster. Note that the sum range varies depending on the logical meaning of each column in the **GLOBAL_STAT_DATABASE** view.
- When you query the **GLOBAL_STAT_DATABASE** view on a DN, the query result is the same as that in **Table 14-106**.

Table 14-106 GLOBAL_STAT_DATABASE columns

Name	Type	Description	Sum Range
datid	oid	Database OID	-
datname	name	Database name	-

Name	Type	Description	Sum Range
numbackends	integer	Number of backends currently connected to this database on the current node. This is the only column in this view that reflects the current state value. All columns return the accumulated value since the last reset.	CN
xact_commit	bigint	Number of transactions in this database that have been committed on the current node	CN
xact_rollback	bigint	Number of transactions in this database that have been rolled back on the current node	CN
blks_read	bigint	Number of disk blocks read in this database on the current node	DN
blks_hit	bigint	Number of disk blocks found in the buffer cache on the current node, that is, the number of blocks hit in the cache. (This only includes hits in the GaussDB(DWS) buffer cache, not in the file system cache.)	DN
tup_returned	bigint	Number of rows returned by queries in this database on the current node	DN
tup_fetched	bigint	Number of rows fetched by queries in this database on the current node	DN
tup_inserted	bigint	Number of rows inserted in this database on the current node	DN
tup_updated	bigint	Number of rows updated in this database on the current node	DN
tup_deleted	bigint	Number of rows deleted from this database on the current node	DN
conflicts	bigint	Number of queries canceled due to database recovery conflicts on the current node (conflicts occurring only on the standby server). For details, see **PG_STAT_DATABASE_CONFLICTS**.	CN and DN

Name	Type	Description	Sum Range
temp_files	bigint	Number of temporary files created by this database on the current node. 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.	DN
temp_bytes	bigint	Size of temporary files written to this database on the current node. All temporary files are counted, regardless of why the temporary file was created, and regardless of the **log_temp_files** setting.	DN
deadlocks	bigint	Number of deadlocks in this database on the current node	CN and DN
blk_read_time	double precision	Time spent reading data file blocks by backends in this database on the current node, in milliseconds	DN
blk_write_time	double precision	Time spent writing into data file blocks by backends in this database on the current node, in milliseconds	DN
stats_reset	timestamp with time zone	Time when the database statistics are reset on the current node	-

14.3.48 GLOBAL_TABLE_CHANGE_STAT

GLOBAL_TABLE_CHANGE_STAT displays the changes of all tables (excluding foreign tables) in the current database. The value of each column that indicates the number of times is the accumulated value since the instance was started.

Table 14-107 GLOBAL_TABLE_CHANGE_STAT columns

Name	Type	Description
schemaname	name	Namespace of a table
relname	name	Table name
last_vacuum	timestamp with time zone	Time when the last **VACUUM** operation is performed manually

Name	Type	Description
vacuum_count	bigint	Number of times of manually performing the **VACUUM** operation. The value is the sum of the number of times on each CN.
last_autovacuum	timestamp with time zone	Time when the last **VACUUM** operation is performed automatically
autovacuum_count	bigint	Number of times of automatically performing the **VACUUM** operation. The value is the sum of the number of times on each CN.
last_analyze	timestamp with time zone	Time when the **ANALYZE** operation is performed (both manually and automatically)
analyze_count	bigint	Number of times of performing the **ANALYZE** operation (both manually and automatically). The **ANALYZE** operation is performed on all CNs at the same time. Therefore, the value of this column is the maximum value on all CNs.
last_autoanalyze	timestamp with time zone	Time when the last **ANALYZE** operation is performed automatically
autoanalyze_count	bigint	Number of times of automatically performing the **ANALYZE** operation. The value is the sum of the number of times on each CN.
last_change	bigint	Time when the last modification (**INSERT**, **UPDATE**, or **DELETE**) is performed

14.3.49 GLOBAL_TABLE_STAT

GLOBAL_TABLE_STAT displays statistics about all tables (excluding foreign tables) in the current database. The values of **live_tuples** and **dead_tuples** are real-time values, and the values of other statistical columns are accumulated values since the instance was started.

Table 14-108 GLOBAL_TABLE_STAT columns

Name	Type	Description
schemaname	name	Namespace of a table
relname	name	Table name

Name	Type	Description
distribute_mode	char	Distribution mode of a table. The meaning of this column is the same as that of the **pclocatortype** column in the **pgxc_class** system catalog.
seq_scan	bigint	Number of sequential scans. It is counted only for row-store tables. For a partitioned table, the sum of the number of scans of each partition is displayed.
seq_tuple_read	bigint	Number of rows scanned in sequence. It is counted only for row-store tables.
index_scan	bigint	Number of index scans. It is counted only for row-store tables.
index_tuple_read	bigint	Number of rows scanned by the index. It is counted only for row-store tables.
tuple_inserted	bigint	Number of rows inserted. For a replication table, the maximum value of each node is displayed. For a distribution table, the sum of all nodes is displayed.
tuple_updated	bigint	Number of rows updated. For a replication table, the maximum value of each node is displayed. For a distribution table, the sum of all nodes is displayed.
tuple_deleted	bigint	Number of rows deleted. For a replication table, the maximum value of each node is displayed. For a distribution table, the sum of all nodes is displayed.
tuple_hot_updated	bigint	Number of rows with HOT updates. For a replication table, the maximum value of each node is displayed. For a distribution table, the sum of all nodes is displayed.
live_tuples	bigint	Number of live tuples. The maximum value of each node is displayed. For a distribution table, the sum of all nodes is displayed. This indicator applies only to row-store tables.
dead_tuples	bigint	Number of dead tuples. The maximum value of each node is displayed. For a distribution table, the sum of all nodes is displayed. This indicator applies only to row-store tables.

14.3.50 GLOBAL_WORKLOAD_SQL_COUNT

GLOBAL_WORKLOAD_SQL_COUNT displays statistics on the number of SQL statements executed in all workload Cgroups in a cluster, including the number of

SELECT, **UPDATE**, **INSERT**, and **DELETE** statements and the number of DDL, DML, and DCL statements.

Table 14-109 GLOBAL_WORKLOAD_SQL_COUNT columns

Name	Type	Description
workload	name	Workload Cgroup 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

14.3.51 GLOBAL_WORKLOAD_SQL_ELAPSE_TIME

GLOBAL_WORKLOAD_SQL_ELAPSE_TIME displays statistics on the response time of SQL statements in all workload Cgroups in a cluster, including the maximum, minimum, average, and total response time of **SELECT**, **UPDATE**, **INSERT**, and **DELETE** statements. The unit is microsecond.

Table 14-110 GLOBAL_WORKLOAD_SQL_ELAPSE_TIME columns

Name	Type	Description
workload	name	Workload Cgroup name
total_select_elapse	bigint	Total response time of **SELECT**
max_select_elapse	bigint	Maximum response time of **SELECT**
min_select_elapse	bigint	Minimum response time of **SELECT**
avg_select_elapse	bigint	Average response time of **SELECT**

Name	Type	Description
total_update_elapse	bigint	Total response time of **UPDATE**
max_update_elapse	bigint	Maximum response time of **UPDATE**
min_update_elapse	bigint	Minimum response time of **UPDATE**
avg_update_elapse	bigint	Average response time of **UPDATE**
total_insert_elapse	bigint	Total response time of **INSERT**
max_insert_elapse	bigint	Maximum response time of **INSERT**
min_insert_elapse	bigint	Minimum response time of **INSERT**
avg_insert_elapse	bigint	Average response time of **INSERT**
total_delete_elapse	bigint	Total response time of **DELETE**
max_delete_elapse	bigint	Maximum response time of **DELETE**
min_delete_elapse	bigint	Minimum response time of **DELETE**
avg_delete_elapse	bigint	Average response time of **DELETE**

14.3.52 GLOBAL_WORKLOAD_TRANSACTION

GLOBAL_WORKLOAD_TRANSACTION provides the total transaction information about workload Cgroups on all CNs in the cluster. This view is accessible only to users with system administrator rights. It is valid only when the real-time resource monitoring function is enabled, that is, **enable_resource_track** is **on**.

Table 14-111 GLOBAL_WORKLOAD_TRANSACTION columns

Name	Type	Description
workload	name	Workload Cgroup name
commit_counter	bigint	Total number of submission times on each CN
rollback_counter	bigint	Total number of rollback times on each CN

Name	Type	Description
resp_min	bigint	Minimum response time of the cluster
resp_max	bigint	Maximum response time of the cluster
resp_avg	bigint	Average response time on each CN
resp_total	bigint	Total response time on each CN

14.3.53 GS_ALL_CONTROL_GROUP_INFO

GS_ALL_CONTROL_GROUP_INFO displays all Cgroup information in a database.

Table 14-112 GS_ALL_CONTROL_GROUP_INFO columns

Name	Type	Description
name	text	Name of the Cgroup
type	text	Type of the Cgroup
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 a Cgroup
limits	bigint	Limit of CPUs allocated to a Cgroup
wdlevel	bigint	Workload Cgroup level
cpucores	text	Usage of CPU cores in a Cgroup

14.3.54 GS_CLUSTER_RESOURCE_INFO

GS_CLUSTER_RESOURCE_INFO displays a DN resource summary.

Table 14-113 GS_CLUSTER_RESOURCE_INFO columns

Name	Type	Description
min_mem_util	integer	Minimum memory usage of a DN
max_mem_util	integer	Maximum memory usage of a DN
min_cpu_util	integer	Minimum CPU usage of a DN

Name	Type	Description
max_cpu_util	integer	Maximum CPU usage of a DN
min_io_util	integer	Minimum I/O usage of a DN
max_io_util	integer	Maximum I/O usage of a DN
used_mem_rate	integer	Maximum physical memory usage

14.3.55 GS_COLUMN_TABLE_IO_STAT

GS_COLUMN_TABLE_IO_STAT displays the I/O of all column-store tables of the database on the current node. The value of each statistical column is the accumulated value since the instance was started.

Table 14-114 GS_COLUMN_TABLE_IO_STAT columns

Name	Type	Description
schemaname	name	Namespace of a table
relname	name	Table name
heap_read	bigint	Number of blocks logically read in the heap
heap_hit	bigint	Number of block hits in the heap
idx_read	bigint	Number of blocks logically read in the index
idx_hit	bigint	Number of block hits in the index
cu_read	bigint	Number of logical reads in the Compression Unit
cu_hit	bigint	Number of hits in the Compression Unit
cidx_read	bigint	Number of indexes logically read in the Compression Unit
cidx_hit	bigint	Number of index hits in the Compression Unit

14.3.56 GS_INSTR_UNIQUE_SQL

Unique SQL Definition

The database parses each received SQL text string and generates an internal parsing tree. The database traverses the parsing tree and ignores constant values in the parsing tree. In this case, an integer value is calculated using a certain algorithm. This integer is used as the Unique SQL ID to uniquely identify this type of SQL. SQL statements with the same Unique SQL ID are called Unique SQL statements.

Examples

Assume that the user enters the following SQL statements in sequence:

```
select * from t1 where id = 1;
select * from t1 where id = 2;
```

The statistics of the two SQL statements are aggregated to the same Unique SQL statement.

```
select * from t1 where id = ?;
```

GS_INSTR_UNIQUE_SQL View

The **GS_INSTR_UNIQUE_SQL** view displays the execution information about the Unique SQL statements collected by the current node, including:

- Unique SQL ID and normalized SQL text string. The normalized SQL text is described in **Examples**. Generally, constant values are ignored during Unique SQL ID calculation in DML statements. However, constant values cannot be ignored in DDL, DCL, and parameter setting statements.
- Number of execution times (number of successful execution times) and response time (SQL execution time in the database, including the maximum, minimum, and total time)
- Cache/IO information, including the number of physical reads and logical reads of a block. Only information about successfully executed SQL statements on each DN is collected. The statistical value is related to factors such as the amount of data processed during query execution, used memory, whether the query is executed for multiple times, memory management policy, and whether there are other concurrent queries. The statistical value reflects the number of physical reads and logical reads of the buffer block in the entire query execution process. The statistical value may vary according to the execution time.
- Row activities, such as the number of returned rows, updated rows, inserted rows, deleted rows, sequentially scanned rows, and randomly scanned rows in the result set of the **SELECT** statement. Except that the number of rows returned by the result set is the same as the number of rows in the result set of the **SELECT** statement and is recorded only on the CN, the activity information of other rows is recorded on the DN. The statistical value reflects the row activities during the entire query execution process, including scanning and modifying related system tables, metadata tables, and data tables. The value of this parameter is related to the data volume and related parameter settings. That is, the statistical value is greater than or equal to the scanning and modification times of actual data tables.
- Time distribution, including DB_TIME/CPU_TIME/EXECUTION_TIME/PARSE_TIME/PLAN_TIME/REWRITE_TIME/PL_EXECUTION_TIME/PL_COMPILATION_TIME/NET_SEND_TIME/DATA_IO_TIME. For details, see **Table 14-115**. The information is collected on both CNs and DNs and is displayed during view query.
- Number of soft and hard parsing times, such as the number of soft parsing times (cache plan) and hard parsing times (generation plan). If the cache

plan is executed this time, the number of soft parsing times increases by 1. If the generation plan is regenerated this time, the number of hard parsing times increases by 1. This number is counted on both CNs and DNs and is displayed during view query.

The Unique SQL statistics function has the following restrictions:

- Detailed statistics are displayed only for successfully executed SQL statements. Otherwise, only query, node, and user information are recorded.
- If the Unique SQL statistics collection function is enabled, the CN collects statistics on all received queries, including tool and user queries.
- If an SQL statement contains multiple SQL statements or similar stored procedures, a Unique SQL statement is generated for the outermost SQL statement. The statistics of all sub-SQL statements are summarized to the Unique SQL record.
- The response time statistics of Unique SQL does not include the time of the **NET_SEND_TIME** phase. Therefore, there is no comparison between **EXECUTION_TIME** and **elapse_time**.
- **parse_time** of clauses cannot be calculated for **begin;...;commit** and similar transaction blocks.

When a common user accesses the **GS_INSTR_UNIQUE_SQL** view, only the Unique SQL information about the user is displayed. When an administrator accesses the **GS_INSTR_UNIQUE_SQL** view, all Unique SQL information about the current node is displayed. The **GS_INSTR_UNIQUE_SQL** view can be queried on both CNs and DNs. The DN displays the Unique SQL statistics of the local node, and the CN displays the complete Unique SQL statistics of the local node. That is, the CN collects the Unique SQL execution information of the CN from other CNs and DNs and displays the information. You can query the **GS_INSTR_UNIQUE_SQL** view to locate the Top SQL statements that consume different resources, providing a basis for cluster tuning and maintenance.

The GUC parameter **instr_unique_sql_timeout** specifies the timeout interval of the Unique SQL statement (in hours). The background thread checks all Unique SQL statements every hour and deletes the Unique SQL statements whose **last_time** is **instr_unique_sql_timeout** hours ago.

Table 14-115 GS_INSTR_UNIQUE_SQL columns

Name	Type	Description
node_name	name	Name of the CN that receives SQL statements
node_id	integer	Node ID, which is the same as the value of **node_id** in the **pgxc_node** table
user_name	name	Username
user_id	oid	User ID

Name	Type	Description
unique_sql_id	bigint	Normalized Unique SQL ID
query	text	Normalized SQL text
n_calls	bigint	Number of successful execution times
min_elapse_time	bigint	Minimum running time of the SQL statement in the database (unit: μs)
max_elapse_time	bigint	Maximum running time of SQL statements in the database (unit: μs)
total_elapse_time	bigint	Total running time of SQL statements in the database (unit: μs)
n_returned_rows	bigint	Row activity - Number of rows in the result set returned by the **SELECT** statement
n_tuples_fetched	bigint	Row activity - Randomly scan rows (column-store tables/foreign tables are not counted.)
n_tuples_returned	bigint	Row activity - Sequential scan rows (Column-store tables/foreign tables are not counted.)
n_tuples_inserted	bigint	Row activity - Inserted rows
n_tuples_updated	bigint	Row activity - Updated rows
n_tuples_deleted	bigint	Row activity - Deleted rows
n_blocks_fetched	bigint	Block access times of the buffer, that is, physical read/I/O
n_blocks_hit	bigint	Block hits of the buffer, that is, logical read/ cache
n_soft_parse	bigint	Number of soft parsing times (cache plan)

Name	Type	Description
n_hard_parse	bigint	Number of hard parsing times (generation plan)
db_time	bigint	Valid DB execution time, including the waiting time and network sending time. If multiple threads are involved in query execution, the value of **DB_TIME** is the sum of **DB_TIME** of multiple threads (unit: µs).
cpu_time	bigint	CPU execution time, excluding the sleep time (unit: µs)
execution_time	bigint	SQL execution time in the query executor, DDL statements, and statements (such as Copy statements) that are not executed by the executor are not counted (unit: µs).
parse_time	bigint	SQL parsing time (unit: µs)
plan_time	bigint	SQL generation plan time (unit: µs)
rewrite_time	bigint	SQL rewriting time (unit: µs)
pl_execution_time	bigint	Execution time of the plpgsql procedural language function (unit: µs)
pl_compilation_time	bigint	Compilation time of the plpgsql procedural language function (unit: µs)
net_send_time	bigint	Network time, including the time spent by the CN in sending data to the client and the time spent by the DN in sending data to the CN (unit: µs)

Name	Type	Description
data_io_time	bigint	File I/O time (unit: μs)
first_time	timestamp with time zone	Time of the first SQL statement execution
last_time	timestamp with time zone	Time of the last SQL statement execution

14.3.57 GS_NODE_STAT_RESET_TIME

GS_NODE_STAT_RESET_TIME provides the statistics reset time of the current node and returns a timestamp with the time zone.

For details, see the **get_node_stat_reset_time()** function.

NOTE

When an instance is running, its statistics keep rising. In the following cases, the statistical values in the memory will be reset to **0**:

- The instance is restarted or a cluster switchover occurs.
- The database is deleted.
- A reset operation is performed. For example, the statistics counter in the database is reset using the **pgstat_recv_resetcounter** function or the Unique SQL statements are cleared using the **reset_instr_unique_sql** function.

If any of the preceding events occurs, GaussDB(DWS) will record the time when the statistics are reset. You can query the time using the **get_node_stat_reset_time** function.

14.3.58 GS_REL_IOSTAT

GS_REL_IOSTAT displays disk I/O statistics on the current node. In the current version, only one page is read or written in each read or write operation. Therefore, the number of read/write times is the same as the number of pages.

Table 14-116 GS_REL_IOSTAT columns

Name	Type	Description
phyrds	bigint	Number of disk reads
phywrts	bigint	Number of disk writes
phyblkrd	bigint	Number of read pages
phyblkwrt	bigint	Number of written pages

14.3.59 GS_RESPOOL_RUNTIME_INFO

GS_RESPOOL_RUNTIME_INFO displays information about the running of jobs in all resource pools on the current CN.

Table 14-117 GS_RESPOOL_RUNTIME_INFO columns

Name	Type	Description
nodegroup	name	Name of the logical cluster of the resource pool. The default value is **installation**.
rpname	name	Resource pool name
ref_count	int	Number of jobs referenced by resource pools. The number is counted regardless of whether a job is controlled by a resource pool.
fast_run	int	Number of running jobs in the fast lane of the resource pool
fast_wait	int	Number of jobs queued in the fast lane of the resource pool
slow_run	int	Number of running jobs in the slow lane of the resource pool
slow_wait	int	Number of jobs queued in the slow lane of the resource pool

14.3.60 GS_RESPOOL_RESOURCE_INFO

GS_RESPOOL_RESOURCE_INFO displays job running information about all resource pools on a CN and the information about resource pool usage of an instance (CN/DN).

NOTE

On a DN, it only displays the monitoring information of the logical cluster that the DN belongs to.

Table 14-118 GS_RESPOOL_RESOURCE_INFO columns

Name	Type	Description
nodegroup	name	Name of the logical cluster of the resource pool. The default value is **installation**.
rpname	name	Resource pool name
cgroup	name	Name of the Cgroup associated with the resource pool
ref_count	int	Number of jobs referenced by the resource pool. The number is counted regardless of whether the job is controlled by the resource pool. This parameter is valid only on CNs.

Name	Type	Description
fast_run	int	Number of running jobs in the fast lane of the resource pool. This parameter is valid only on CNs.
fast_wait	int	Number of jobs queued in the fast lane of the resource pool. This parameter is valid only on CNs.
fast_limit	int	Limit on the number of concurrent fast lane jobs in the resource pool. This parameter is valid only on CNs.
slow_run	int	Number of running jobs in the slow lane of the resource pool. This parameter is valid only on CNs.
slow_wait	int	Number of jobs queued in the slow lane of the resource pool. This parameter is valid only on CNs.
slow_limit	int	Limit on the number of concurrent slow lane jobs in the resource pool. This parameter is valid only on CNs.
used_cpu	double	Average number of used CPUs of the resource pool in a 5s monitoring period. The value is accurate to two decimal places. ● On a DN, it indicates the number of CPUs used by the resource pool on the current DN. ● On a CN, it indicates the total CPU usage of resource pools on all DNs.
cpu_limit	int	Specifies the cap of available CPUs in the resource pool. If the CPU time limit is specified, this parameter indicates the available CPUs for GaussDB(DWS). If the CPU usage limit is specified, this parameter indicates the available CPUs for associated Cgroups. ● On a DN, it indicates the upper limit of available CPUs for the resource pool on the current DN. ● On a CN, it indicates the total upper limit of available CPUs for resource pools on all DNs.

Name	Type	Description
used_mem	int	Memory size used by the resource pool (unit: MB) ● On a DN, it indicates the memory usage of the resource pool on the current DN. ● On a CN, it indicates the total memory usage of resource pools on all DNs.
estimate_mem	int	Estimated memory used by the jobs running in the resource pool on the current CN. This parameter is valid only on CNs.
mem_limit	int	Upper limit of available memory for resource pools, in MB. ● On a DN, it indicates the upper limit of available memory for the resource pool on the current DN. ● On a CN, it indicates the total upper limit of available memory for resource pools on all DNs.
read_kbytes	bigint	Number of logical read bytes in the resource pool within a 5s monitoring period (unit: KB). ● On a DN, it indicates the number of logical read bytes in the resource pool on the current DN. ● On a CN, it indicates the total logical read bytes of resource pools on all DNs.
write_kbytes	bigint	Number of logical write bytes in the resource pool within a 5s monitoring period (unit: KB). ● On a DN, it indicates the number of logical write bytes in the resource pool on the current DN. ● On a CN, it indicates the total logical write bytes of resource pools on all DNs.
read_counts	bigint	Number of logical reads in the resource pool within a 5s monitoring period. ● On a DN, it indicates the number of logical reads in the resource pool on the current DN. ● On a CN, it indicates the total number of logical reads in resource pools on all DNs.

Name	Type	Description
write_counts	bigint	Number of logical writes in the resource pool within a 5s monitoring period. ● On a DN, it indicates the number of logical writes in the resource pool on the current DN. ● On a CN, it indicates the total number of logical writes in resource pools on all DNs.
read_speed	double	Average rate of logical reads of the resource pool in a 5s monitoring period. ● On a DN, it indicates the logical read rate of the resource pool on the current DN. ● On a CN, it indicates the overall logical read rate of resource pools on all DNs.
write_speed	double	Average rate of logical writes of the resource pool in a 5s monitoring period. ● On a DN, it indicates the logical write rate of the resource pool on the current DN. ● On a CN, it indicates the overall logical write rate of resource pools on all DNs.

14.3.61 GS_ROW_TABLE_IO_STAT

GS_ROW_TABLE_IO_STAT displays the I/O of all row-store tables of the database on the current node. The value of each statistical column is the accumulated value since the instance was started.

Table 14-119 GS_ROW_TABLE_IO_STAT columns

Name	Type	Description
schemaname	name	Namespace of a table
relname	name	Table name
heap_read	bigint	Number of blocks logically read in the heap
heap_hit	bigint	Number of block hits in the heap
idx_read	bigint	Number of blocks logically read in the index
idx_hit	bigint	Number of block hits in the index
toast_read	bigint	Number of blocks logically read in the **TOAST** table
toast_hit	bigint	Number of block hits in the **TOAST** table

Name	Type	Description
tidx_read	bigint	Number of indexes logically read in the **TOAST** table
tidx_hit	bigint	Number of index hits in the **TOAST** table

14.3.62 GS_SESSION_CPU_STATISTICS

GS_SESSION_CPU_STATISTICS displays load management information about CPU usage of ongoing complex jobs executed by the current user.

Table 14-120 GS_SESSION_CPU_STATISTICS columns

Name	Type	Description
datid	oid	OID of the database this backend is connected to
usename	name	Name of the user logging in to the backend
pid	bigint	ID of a backend process
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 all DNs. The unit is ms.
max_cpu_time	bigint	Maximum CPU time of the statement across all DNs. The unit is ms.
total_cpu_time	bigint	Total CPU time of the statement across all DNs. The unit is ms.
query	text	Statement that is being executed
node_group	text	Logical cluster of the user running the statement

14.3.63 GS_SESSION_MEMORY_STATISTICS

GS_SESSION_MEMORY_STATISTICS displays load management information about memory usage of ongoing complex jobs executed by the current user.

Table 14-121 GS_SESSION_MEMORY_STATISTICS columns

Name	Type	Description
datid	oid	OID of the database the backend is connected to
usename	name	Name of the user logged in to the backend
pid	bigint	ID of the backend thread
start_time	timestamp with time zone	Time when the statement starts to be executed
min_peak_memory	integer	Minimum memory peak of a statement across all DNs, in MB
max_peak_memory	integer	Maximum memory peak of a statement across all DNs, in MB
spill_info	text	Statement spill information on all DNs. ● **None** indicates that the statement has not been flushed to disks on any DNs. ● **All** indicates that the statement has been spilled to disks on every DN. ● **[*a*:*b*]** indicates that the statement has been spilled to disks on *a* of *b* DNs.
query	text	Statement that is being executed
node_group	text	Logical cluster of the user running the statement

14.3.64 GS_SQL_COUNT

GS_SQL_COUNT displays statistics about the five types of statements (**SELECT**, **INSERT**, **UPDATE**, **DELETE**, and **MERGE INTO**) executed on the current node of the database, including the number of execution times, response time (the maximum, minimum, average, and total response time of the other four types of statements except the **MERGE INTO** statement, in microseconds), and the number of execution times of **DDL**, **DML**, and **DCL statements**.

The classification of **DDL**, **DML**, and **DCL** statements in the **GS_SQL_COUNT** view is slightly different from that of the SQL syntaxt. The details are as follows:

- User-related statements, such as **CREATE**/**ALTER**/**DROP USER** and **CREATE**/**ALTER**/**DROP ROLE**, are of the DCL type.
- Transaction-related statements such as **BEGIN**/**COMMIT**/**SET CONSTRAINTS**/**ROLLBACK**/**SAVEPOINT**/**START** are of the DCL type.
- **ALTER SYSTEM KILL SESSION** is equivalent to the **SELECT pg_terminate_backend()** statement and is of the DML type.

The classification of other statements is similar to the definition in the SQL syntax.

When a common user queries the **GS_SQL_COUNT** view, only the statistics of this user in the current node can be viewed. When a user with the administrator permissions queries the **GS_SQL_COUNT** view, the statistics of all users in the current node can be viewed. When the cluster or the node is restarted, the statistics are cleared and the counting restarts. The counting is based on the number of queries received by the node, including the queries performed inside the cluster. Statistics about the **GS_SQL_COUNT** view are collected only on CNs, and SQL statements sent from other CNs are not collected. No result is returned when you query the view on a DN.

Table 14-122 GS_SQL_COUNT columns

Name	Type	Description
node_name	name	Node name
user_name	name	User 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
mergeinto_count	bigint	Number of **MERGE INTO** statements
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
avg_select_elapse	bigint	Average response time of **SELECT** statements
max_select_elapse	bigint	Maximum response time of **SELECT** statements
min_select_elapse	bigint	Minimum response time of **SELECT** statements
total_update_elapse	bigint	Total response time of **UPDATE** statements
avg_update_elapse	bigint	Average response time of **UPDATE** statements
max_update_elapse	bigint	Maximum response time of **UPDATE** statements
min_update_elapse	bigint	Minimum response time of **UPDATE** statements

Name	Type	Description
total_delete_elapse	bigint	Total response time of **DELETE** statements
avg_delete_elapse	bigint	Average response time of **DELETE** statements
max_delete_elapse	bigint	Maximum response time of **DELETE** statements
min_delete_elapse	bigint	Minimum response time of **DELETE** statements
total_insert_elapse	bigint	Total response time of **INSERT** statements
avg_insert_elapse	bigint	Average response time of **INSERT** statements
max_insert_elapse	bigint	Maximum response time of **INSERT** statements
min_insert_elapse	bigint	Minimum response time of **INSERT** statements

14.3.65 GS_STAT_DB_CU

GS_STAT_DB_CU displays CU hits of each database in each node of a cluster. You can clear it using **gs_stat_reset()**.

Table 14-123 GS_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 disk synchronous reads
hdd_asyn_read	bigint	Number of disk asynchronous reads

14.3.66 GS_STAT_SESSION_CU

GS_STAT_SESSION_CU displays the CU hit rate of running sessions on each node in a cluster. This data about a session is cleared when you exit this session or restart the cluster.

Table 14-124 GS_STAT_SESSION_CU columns

Name	Type	Description
node_name1	text	Node name
mem_hit	integer	Number of memory hits
hdd_sync_read	integer	Number of disk synchronous reads
hdd_asyn_read	integer	Number of disk asynchronous reads

14.3.67 GS_TABLE_CHANGE_STAT

GS_TABLE_CHANGE_STAT displays the changes of all tables (excluding foreign tables) of the database on the current node. The value of each column that indicates the number of times is the accumulated value since the instance was started.

Table 14-125 GS_TABLE_CHANGE_STAT columns

Name	Type	Description
schemaname	name	Namespace of a table
relname	name	Table name
last_vacuum	timestamp with time zone	Time when the last **VACUUM** operation is performed manually
vacuum_count	bigint	Number of times of manually performing the **VACUUM** operation
last_autovacuum	timestamp with time zone	Time when the last **VACUUM** operation is performed automatically
autovacuum_count	bigint	Number of times of automatically performing the **VACUUM** operation
last_analyze	timestamp with time zone	Time when the **ANALYZE** operation is performed (both manually and automatically)
analyze_count	bigint	Number of times of performing the **ANALYZE** operation (both manually and automatically)
last_autoanalyze	timestamp with time zone	Time when the last **ANALYZE** operation is performed automatically

Name	Type	Description
autoanalyze_count	bigint	Number of times of automatically performing the **ANALYZE** operation
last_change	bigint	Time when the last modification (**INSERT**, **UPDATE**, or **DELETE**) is performed

14.3.68 GS_TABLE_STAT

GS_TABLE_STAT displays statistics about all tables (excluding foreign tables) of the database on the current node. The values of **live_tuples** and **dead_tuples** are real-time values, and the values of other statistical columns are accumulated values since the instance was started.

Table 14-126 GS_TABLE_STAT columns

Name	Type	Description
schemaname	name	Namespace of a table
relname	name	Table name
seq_scan	bigint	Number of sequential scans. It is counted only for row-store tables. For a partitioned table, the sum of the number of scans of each partition is displayed.
seq_tuple_read	bigint	Number of rows scanned in sequence. It is counted only for row-store tables.
index_scan	bigint	Number of index scans. It is counted only for row-store tables.
index_tuple_read	bigint	Number of rows scanned by the index. It is counted only for row-store tables.
tuple_inserted	bigint	Number of rows inserted.
tuple_updated	bigint	Number of rows updated.
tuple_deleted	bigint	Number of rows deleted.
tuple_hot_updated	bigint	Number of rows with HOT updates.
live_tuples	bigint	Number of live tuples. Query the view on the CN. If **ANALYZE** is executed, the total number of live tuples in the table is displayed. Otherwise, **0** is displayed. This indicator applies only to row-store tables.

Name	Type	Description
dead_tuples	bigint	Number of dead tuples. Query the view on the CN. If **ANALYZE** is executed, the total number of dead tuples in the table is displayed. Otherwise, **0** is displayed. This indicator applies only to row-store tables.

14.3.69 GS_TOTAL_NODEGROUP_MEMORY_DETAIL

GS_TOTAL_NODEGROUP_MEMORY_DETAIL displays statistics about memory usage of the logical cluster that the current database belongs to in the unit of MB.

Table 14-127 GS_TOTAL_NODEGROUP_MEMORY_DETAIL columns

Name	Type	Description
ngname	text	Name of a logical cluster
memorytype	text	Memory type. Its value can be: ● **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 cluster 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

14.3.70 GS_USER_TRANSACTION

GS_USER_TRANSACTION provides transaction information about users on a single CN. The database records the number of times that each user commits and rolls back transactions and the response time of transaction commitment and rollback, in microseconds.

Table 14-128 GS_USER_TRANSACTION columns

Name	Type	Description
usename	name	Username
commit_counter	bigint	Number of the commit times
rollback_counter	bigint	Number of rollbacks
resp_min	bigint	Minimum response time
resp_max	bigint	Maximum response time
resp_avg	bigint	Average response time
resp_total	bigint	Total response time

14.3.71 GS_VIEW_DEPENDENCY

GS_VIEW_DEPENDENCY allows you to query the direct dependencies of all views visible to the current user.

Table 14-129 GS_VIEW_DEPENDENCY columns

Column	Type	Description
objschema	name	View space name
objname	name	View name
refobjschema	name	Name of the space where the dependent object resides
refobjname	name	Name of a dependent object
relobjkind	char	Type of a dependent object ● **r**: table ● **v**: view

14.3.72 GS_VIEW_DEPENDENCY_PATH

GS_VIEW_DEPENDENCY_PATH allows you to query the direct dependencies of all views visible to the current user. If the base table on which the view depends exists and the dependency between views at different levels is normal, you can use this view to query the dependency between views at different levels starting from the base table.

Table 14-130 GS_VIEW_DEPENDENCY_PATH columns

Column	Type	Description
objschema	name	View space name
objname	name	View name
refobjschema	name	Name of the space where the dependent object resides
refobjname	name	Name of a dependent object
path	text	Dependency path

14.3.73 GS_VIEW_INVALID

GS_VIEW_INVALID queries all unavailable views visible to the current user. If the base table, function, or synonym that the view depends on is abnormal, the **validtype** column of the view is displayed as "invalid".

Table 14-131 GS_VIEW_INVALID columns

Column	Type	Description
oid	oid	OID of the view
schemaname	name	View space name
viewname	name	Name of the view
viewowner	name	Owner of the view
definition	text	Definition of the view
validtype	text	View validity flag

14.3.74 GS_WAIT_EVENTS

GS_WAIT_EVENTS displays statistics about waiting status and events on the current node.

The values of statistical columns in this view are accumulated only when the **enable_track_wait_event** GUC parameter is set to **on**. If

enable_track_wait_event is set to **off** during statistics measurement, the statistics will no longer be accumulated, but the existing values are not affected. If **enable_track_wait_event** is **off**, 0 row is returned when this view is queried.

Table 14-132 GS_WAIT_EVENTS columns

Name	Type	Description
nodename	name	Node name
type	text	Event type, which can be **STATUS**, **LOCK_EVENT**, **LWLOCK_EVENT**, or **IO_EVENT**
event	text	Event name. For details, see PG_THREAD_WAIT_STATUS.
wait	bigint	Number of times an event occurs. This column and all the columns below are values accumulated during process running.
failed_wait	bigint	Number of waiting failures. In the current version, this column is used only for counting timeout errors and waiting failures of locks such as **LOCK** and **LWLOCK**.
total_wait_time	bigint	Total duration of the event
avg_wait_time	bigint	Average duration of the event
max_wait_time	bigint	Maximum wait time of the event
min_wait_time	bigint	Minimum wait time of the event

In the current version, for events whose **type** is **LOCK_EVENT**, **LWLOCK_EVENT**, or **IO_EVENT**, the display scope of **GS_WAIT_EVENTS** is the same as that of the corresponding events in the **PG_THREAD_WAIT_STATUS** view.

For events whose **type** is **STATUS**, **GS_WAIT_EVENTS** displays the following waiting status columns. For details, see the **PG_THREAD_WAIT_STATUS** view.

- acquire lwlock
- acquire lock
- wait io
- wait pooler get conn
- wait pooler abort conn
- wait pooler clean conn
- wait transaction sync
- wait wal sync
- wait data sync

- wait producer ready
- create index
- analyze
- vacuum
- vacuum full
- gtm connect
- gtm begin trans
- gtm commit trans
- gtm rollback trans
- gtm create sequence
- gtm alter sequence
- gtm get sequence val
- gtm set sequence val
- gtm drop sequence
- gtm rename sequence

14.3.75 GS_WLM_OPERAROR_INFO

This view displays the execution information about operators in the query statements that have been executed on the current CN. The information comes from the system catalog **dbms_om**. **gs_wlm_operator_info**.

Table 14-133 GS_WLM_OPERATOR_INFO columns

Name	Type	Description
nodename	text	Name of the CN where the statement is executed
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 plan_node_id
start_time	timestamp with time zone	Time when an operator starts to process the first data record
duration	bigint	Total execution time of an operator. The unit is ms.
query_dop	integer	Degree of parallelism (DOP) of the current operator
estimated_rows	bigint	Number of rows estimated by the optimizer

Name	Type	Description
tuple_processed	bigint	Number of elements returned by the current operator
min_peak_memory	integer	Minimum peak memory used by the current operator on all DNs. The unit is MB.
max_peak_memory	integer	Maximum peak memory used by the current operator on all DNs. The unit is MB.
average_peak_memory	integer	Average peak memory used by the current operator on all DNs. The unit is MB.
memory_skew_percent	integer	Memory usage skew of the current operator among DNs
min_spill_size	integer	Minimum spilled data among all DNs when a spill occurs. The unit is MB. Default value: **0**.
max_spill_size	integer	Maximum spilled data among all DNs when a spill occurs. The unit is MB. Default value: **0**.
average_spill_size	integer	Average spilled data among all DNs when a spill occurs. The unit is MB. Default value: **0**.
spill_skew_percent	integer	DN spill skew when a spill occurs
min_cpu_time	bigint	Minimum execution time of the operator on all DNs. The unit is ms.
max_cpu_time	bigint	Maximum execution time of the operator on all DNs. The unit is ms.
total_cpu_time	bigint	Total execution time of the operator on all DNs. The unit is ms.
cpu_skew_percent	integer	Skew of the execution time among DNs.
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

14.3.76 GS_WLM_OPERATOR_HISTORY

GS_WLM_OPERATOR_HISTORY displays the records of operators in jobs that have been executed by the current user on the current CN.

This view is used to query data from the GaussDB(DWS). Data in the GaussDB(DWS) is cleared periodically. If the GUC parameter **enable_resource_record** is set to **on**, records in the view will be dumped to the system catalog **GS_WLM_OPERATOR_INFO** every three minutes and deleted from the view. If **enable_resource_record** is set to **off**, the records will be deleted from the view after the retention period expires. The recorded data is the same as that described in **Table 14-134**.

Table 14-134 GS_WLM_OPERATOR_INFO columns

Name	Type	Description
nodename	text	Name of the CN where the statement is executed
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 plan_node_id
start_time	timestamp with time zone	Time when an operator starts to process the first data record
duration	bigint	Total execution time of an operator. The unit is ms.
query_dop	integer	Degree of parallelism (DOP) of the current operator
estimated_rows	bigint	Number of rows estimated by the optimizer
tuple_processed	bigint	Number of elements returned by the current operator
min_peak_memory	integer	Minimum peak memory used by the current operator on all DNs. The unit is MB.
max_peak_memory	integer	Maximum peak memory used by the current operator on all DNs. The unit is MB.
average_peak_memory	integer	Average peak memory used by the current operator on all DNs. The unit is MB.
memory_skew_percent	integer	Memory usage skew of the current operator among DNs

Name	Type	Description
min_spill_size	integer	Minimum spilled data among all DNs when a spill occurs. The unit is MB. Default value: **0**.
max_spill_size	integer	Maximum spilled data among all DNs when a spill occurs. The unit is MB. Default value: **0**.
average_spill_size	integer	Average spilled data among all DNs when a spill occurs. The unit is MB. Default value: **0**.
spill_skew_percent	integer	DN spill skew when a spill occurs
min_cpu_time	bigint	Minimum execution time of the operator on all DNs. The unit is ms.
max_cpu_time	bigint	Maximum execution time of the operator on all DNs. The unit is ms.
total_cpu_time	bigint	Total execution time of the operator on all DNs. The unit is ms.
cpu_skew_percent	integer	Skew of the execution time among DNs.
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

14.3.77 GS_WLM_OPERATOR_STATISTICS

GS_WLM_OPERATOR_STATISTICS displays the operators of the jobs that are being executed by the current user.

Table 14-135 GS_WLM_OPERATOR_STATISTICS columns

Name	Type	Description
queryid	bigint	Internal query_id used for statement execution
pid	bigint	ID of the backend thread
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 plan_node_id
start_time	timestamp with time zone	Time when an operator starts to process the first data record
duration	bigint	Total execution time of an operator. The unit is ms.
status	text	Execution status of the current operator. Its value can be **finished** or **running**.
query_dop	integer	DOP of the current operator
estimated_rows	bigint	Number of rows estimated by the optimizer
tuple_processe d	bigint	Number of elements returned by the current operator
min_peak_mem ory	integer	Minimum peak memory used by the current operator on all DNs. The unit is MB.
max_peak_me mory	integer	Maximum peak memory used by the current operator on all DNs. The unit is MB.
average_peak_ memory	integer	Average peak memory used by the current operator on all DNs. The unit is MB.
memory_skew_ percent	integer	Memory usage skew of the current operator among DNs
min_spill_size	integer	Minimum spilled data among all DNs when a spill occurs. The default value is **0**. The unit is MB.
max_spill_size	integer	Minimum spilled data among all DNs when a spill occurs. The default value is **0**. The unit is MB.
average_spill_si ze	integer	Average spilled data among all DNs when a spill occurs. The default value is **0**. The unit is MB.
spill_skew_perc ent	integer	DN spill skew when a spill occurs
min_cpu_time	bigint	Minimum execution time of the operator on all DNs. The unit is ms.
max_cpu_time	bigint	Maximum execution time of the operator on all DNs. The unit is ms.
total_cpu_time	bigint	Total execution time of the operator on all DNs. The unit is ms.

Name	Type	Description
cpu_skew_percent	integer	Skew of the execution time among DNs.
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

14.3.78 GS_WLM_SESSION_INFO

This view displays the execution information about the query statements that have been executed on the current CN. The information comes from the system catalog **dbms_om**. **gs_wlm_session_info**.

For details about columns in the view, see **Table 14-136**.

Table 14-136 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 CN where the statement is run
username	text	User name used for connecting to the backend
application_name	text	Name of the application that is connected to the backend
client_addr	inet	IP address of the client connected to this 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.

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 only be non-null 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, or **-1** if a Unix socket is used
query_band	text	Job type, which is specified by the **query_band** parameter. The default value is a null string.
block_time	bigint	Duration that a statement is blocked before being executed, including the statement parsing and optimization duration. The unit is 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. Its value can be **finished** (normal) or **aborted** (abnormal). The statement status here is the execution status of the database server. If the statement is successfully executed on the database server but an error is reported in the result set, the statement status is **finished**.
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. The unit is MB.
min_peak_memory	integer	Minimum memory peak of a statement across all DNs. The unit is MB.
max_peak_memory	integer	Maximum memory peak of a statement across all DNs. The unit is MB.

Name	Type	Description
average_peak_memory	integer	Average memory usage during statement execution. The unit is MB.
memory_skew_percent	integer	Memory usage skew of a statement among DNs.
spill_info	text	Statement spill information on all DNs. **None** indicates that the statement has not been flushed to disks on any DNs. **All** indicates that the statement has been spilled to disks on every DN. **[*a*:*b*]** indicates that 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. The default value is **0**. The unit is MB.
max_spill_size	integer	Minimum spilled data among all DNs when a spill occurs. The default value is **0**. The unit is MB.
average_spill_size	integer	Average spilled data among all DNs when a spill occurs. The default value is **0**. The unit is MB.
spill_skew_percent	integer	DN spill skew when a spill occurs
min_dn_time	bigint	Minimum execution time of a statement across all DNs. The unit is ms.
max_dn_time	bigint	Maximum execution time of a statement across all DNs. The unit is ms.
average_dn_time	bigint	Average execution time of a statement across all DNs. The unit is ms.
dntime_skew_percent	integer	Execution time skew of a statement among DNs.
min_cpu_time	bigint	Minimum CPU time of a statement across all DNs. The unit is ms.
max_cpu_time	bigint	Maximum CPU time of a statement across all DNs. The unit is ms.
total_cpu_time	bigint	Total CPU time of a statement across all DNs. The unit is ms.
cpu_skew_percent	integer	CPU time skew of a statement among DNs.

Name	Type	Description
min_peak_iops	integer	Minimum I/O peak of a statement on all DNs (times/s in column-store tables and 10,000 times/s in row-store tables). This function is not enabled in cluster version 8.1.3. Therefore, you are not advised to refer to this field to analyze memory problems.
max_peak_iops	integer	Maximum I/O peak of a statement on all DNs (times/s in column-store tables and 10,000 times/s in row-store tables). This function is not enabled in cluster version 8.1.3. Therefore, you are not advised to refer to this field to analyze memory problems.
average_peak_iops	integer	Average I/O peak of a statement on all DNs (times/s in column-store tables and 10,000 times/s in row-store tables). This function is not enabled in cluster version 8.1.3. Therefore, you are not advised to refer to this field to analyze memory problems.
iops_skew_percent	integer	I/O skew of a statement among DNs. This function is not enabled in the 8.1.3 cluster. You are not advised to refer to this field to analyze memory problems.
warning	text	Warning. The following warnings and warnings related to SQL self-diagnosis tuning 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 cluster of the user running the statement
pid	bigint	PID of the backend thread of the statement
lane	text	Fast/Slow lane where the statement is executed
unique_sql_id	bigint	ID of the normalized unique SQL.

14.3.79 GS_WLM_SESSION_HISTORY

GS_WLM_SESSION_HISTORY displays load management information about a completed job executed by the current user on the current CN. The view is used by Database Manager to query data from GaussDB(DWS). The view returns the data queried from the **GS_WLM_SESSION_INFO** table within three minutes only when the GUC parameter enable_resource_track is set to **on**.

Table 14-137 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 CN where the statement is run
username	text	User name used for connecting to the backend
application_name	text	Name of the application that is connected to the backend
client_addr	inet	IP address of the client connected to this 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 only be non-null 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, or **-1** if a Unix socket is used
query_band	text	Job type, which is specified by the **query_band** parameter. The default value is a null string.
block_time	bigint	Duration that a statement is blocked before being executed, including the statement parsing and optimization duration. The unit is ms.
start_time	timestamp with time zone	Time when the statement starts to be run

Name	Type	Description
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. Its value can be **finished** (normal) or **aborted** (abnormal). The statement status here is the execution status of the database server. If the statement is successfully executed on the database server but an error is reported in the result set, the statement status is **finished**.
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. The unit is MB.
min_peak_memory	integer	Minimum memory peak of a statement across all DNs. The unit is MB.
max_peak_memory	integer	Maximum memory peak of a statement across all DNs. The unit is MB.
average_peak_memory	integer	Average memory usage during statement execution. The unit is MB.
memory_skew_percent	integer	Memory usage skew of a statement among DNs.
spill_info	text	Statement spill information on all DNs. **None** indicates that the statement has not been flushed to disks on any DNs. **All** indicates that the statement has been spilled to disks on every DN. **[*a*:*b*]** indicates that 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. The default value is **0**. The unit is MB.
max_spill_size	integer	Minimum spilled data among all DNs when a spill occurs. The default value is **0**. The unit is MB.

Name	Type	Description
average_spill_size	integer	Average spilled data among all DNs when a spill occurs. The default value is **0**. The unit is MB.
spill_skew_percent	integer	DN spill skew when a spill occurs
min_dn_time	bigint	Minimum execution time of a statement across all DNs. The unit is ms.
max_dn_time	bigint	Maximum execution time of a statement across all DNs. The unit is ms.
average_dn_time	bigint	Average execution time of a statement across all DNs. The unit is ms.
dntime_skew_percent	integer	Execution time skew of a statement among DNs.
min_cpu_time	bigint	Minimum CPU time of a statement across all DNs. The unit is ms.
max_cpu_time	bigint	Maximum CPU time of a statement across all DNs. The unit is ms.
total_cpu_time	bigint	Total CPU time of a statement across all DNs. The unit is ms.
cpu_skew_percent	integer	CPU time skew of a statement among DNs.
min_peak_iops	integer	Minimum I/O peak of a statement on all DNs (times/s in column-store tables and 10,000 times/s in row-store tables). This function is not enabled in cluster version 8.1.3. Therefore, you are not advised to refer to this field to analyze memory problems.
max_peak_iops	integer	Maximum I/O peak of a statement on all DNs (times/s in column-store tables and 10,000 times/s in row-store tables). This function is not enabled in cluster version 8.1.3. Therefore, you are not advised to refer to this field to analyze memory problems.
average_peak_iops	integer	Average I/O peak of a statement on all DNs (times/s in column-store tables and 10,000 times/s in row-store tables). This function is not enabled in cluster version 8.1.3. Therefore, you are not advised to refer to this field to analyze memory problems.

Name	Type	Description
iops_skew_percent	integer	I/O skew of a statement among DNs. This function is not enabled in the 8.1.3 cluster. You are not advised to refer to this field to analyze memory problems.
warning	text	Warning. The following warnings and warnings related to SQL self-diagnosis tuning 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 cluster of the user running the statement
pid	bigint	PID of the backend thread of the statement
lane	text	Fast/Slow lane where the statement is executed
unique_sql_id	bigint	ID of the normalized unique SQL.

14.3.80 GS_WLM_SESSION_STATISTICS

GS_WLM_SESSION_STATISTICS displays load management information about jobs being executed by the current user on the current CN.

Table 14-138 GS_WLM_SESSION_STATISTICS columns

Name	Type	Description
datid	oid	OID of the database this backend is connected to
dbname	name	Name of the database the backend is connected to
schemaname	text	Schema name
nodename	text	Name of the CN where the statement is executed

Name	Type	Description
username	name	User name used for connecting to the backend
application_name	text	Name of the application that is connected to the backend
client_addr	inet	IP address of the client connected to this 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 only be non-null 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, or **-1** if a Unix socket is used
query_band	text	Job type, which is specified by the GUC parameter **query_band** parameter. The default value is a null string.
pid	bigint	Process ID of the backend
block_time	bigint	Block time before the statement is executed. The unit is ms.
start_time	timestamp with time zone	Time when the statement starts to be executed
duration	bigint	For how long a statement has been executing. The unit is ms.
estimate_total_time	bigint	Estimated execution time of a statement. The unit is ms.
estimate_left_time	bigint	Estimated remaining time of statement execution. The unit is ms.
enqueue	text	Workload management resource status
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 a statement across all DNs. The unit is MB.

Name	Type	Description
max_peak_memory	integer	Maximum memory peak of a statement across all DNs. The unit is MB.
average_peak_memory	integer	Average memory usage during statement execution. The unit is MB.
memory_skew_percent	integer	Memory usage skew of a statement among DNs.
spill_info	text	Statement spill information on all DNs. **None** indicates that the statement has not been flushed to disks on any DNs. **All** indicates that the statement has been spilled to disks on every DN. **[*a*:*b*]** indicates that 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. The unit is MB. Default value: **0**.
max_spill_size	integer	Maximum spilled data among all DNs when a spill occurs. The unit is MB. Default value: **0**.
average_spill_size	integer	Average spilled data among all DNs when a spill occurs. The unit is MB. Default value: **0**.
spill_skew_percent	integer	DN spill skew when a spill occurs
min_dn_time	bigint	Minimum execution time of a statement across all DNs. The unit is ms.
max_dn_time	bigint	Maximum execution time of a statement across all DNs. The unit is ms.
average_dn_time	bigint	Average execution time of a statement across all DNs. The unit is ms.
dntime_skew_percent	integer	Execution time skew of a statement among DNs.
min_cpu_time	bigint	Minimum CPU time of a statement across all DNs. The unit is ms.
max_cpu_time	bigint	Maximum CPU time of a statement across all DNs. The unit is ms.
total_cpu_time	bigint	Total CPU time of a statement across all DNs. The unit is ms.
cpu_skew_percent	integer	CPU time skew of a statement among DNs.

Name	Type	Description
min_peak_iops	integer	Minimum I/O peak of a statement on all DNs (times/s in column-store tables and 10,000 times/s in row-store tables). This function is not enabled in cluster version 8.1.3. Therefore, you are not advised to refer to this field to analyze memory problems.
max_peak_iops	integer	Maximum I/O peak of a statement on all DNs (times/s in column-store tables and 10,000 times/s in row-store tables). This function is not enabled in cluster version 8.1.3. Therefore, you are not advised to refer to this field to analyze memory problems.
average_peak_iops	integer	Average I/O peak of a statement on all DNs (times/s in column-store tables and 10,000 times/s in row-store tables). This function is not enabled in cluster version 8.1.3. Therefore, you are not advised to refer to this field to analyze memory problems.
iops_skew_percent	integer	I/O skew across DNs. This function is not enabled in the 8.1.3 cluster version. You are not advised to analyze memory problems by referring to this field.
warning	text	Warning. The following warnings and warnings related to SQL self-diagnosis tuning 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
query_plan	text	Execution plan of a statement
node_group	text	Logical cluster of the user running the statement

14.3.81 GS_WLM_SQL_ALLOW

The **GS_WLM_SQL_ALLOW** view displays the configured resource management SQL whitelist.

The whitelist contains:

- Default SQL whitelist of the system.
- SQL whitelist specified by the GUC parameter **wlm_sql_allow_list**.

14.3.82 GS_WORKLOAD_SQL_COUNT

GS_WORKLOAD_SQL_COUNT displays statistics on the number of SQL statements executed in workload Cgroups on the current node, including the number of **SELECT**, **UPDATE**, **INSERT**, and **DELETE** statements and the number of DDL, DML, and DCL statements.

Table 14-139 GS_WORKLOAD_SQL_COUNT columns

Name	Type	Description
workload	name	Workload Cgroup 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

14.3.83 GS_WORKLOAD_SQL_ELAPSE_TIME

GS_WORKLOAD_SQL_ELAPSE_TIME displays statistics on the response time of SQL statements in workload Cgroups on the current node, including the maximum, minimum, average, and total response time of **SELECT**, **UPDATE**, **INSERT**, and **DELETE** statements. The unit is microsecond.

Table 14-140 GS_WORKLOAD_SQL_ELAPSE_TIME columns

Name	Type	Description
workload	name	Workload Cgroup name

Name	Type	Description
total_select_elapse	bigint	Total response time of **SELECT** statements
max_select_elapse	bigint	Maximum response time of **SELECT** statements
min_select_elapse	bigint	Minimum response time of **SELECT** statements
avg_select_elapse	bigint	Average response time of **SELECT** statements
total_update_elapse	bigint	Total response time of **UPDATE** statements
max_update_elapse	bigint	Maximum response time of **UPDATE** statements
min_update_elapse	bigint	Minimum response time of **UPDATE** statements
avg_update_elapse	bigint	Average response time of **UPDATE** statements
total_insert_elapse	bigint	Total response time of **INSERT** statements
max_insert_elapse	bigint	Maximum response time of **INSERT** statements
min_insert_elapse	bigint	Minimum response time of **INSERT** statements
avg_insert_elapse	bigint	Average response time of **INSERT** statements
total_delete_elapse	bigint	Total response time of **DELETE** statements
max_delete_elapse	bigint	Maximum response time of **DELETE** statements
min_delete_elapse	bigint	Minimum response time of **DELETE** statements
avg_delete_elapse	bigint	Average response time of **DELETE** statements

14.3.84 GS_WORKLOAD_TRANSACTION

GS_WORKLOAD_TRANSACTION provides transaction information about workload cgroups on a single CN. The database records the number of times that each workload Cgroup commits and rolls back transactions and the response time of transaction commitment and rollback, in microseconds.

Table 14-141 GS_WORKLOAD_TRANSACTION columns

Name	Type	Description
workload	name	Workload Cgroup name
commit_counter	bigint	Number of the commit times
rollback_counter	bigint	Number of rollbacks
resp_min	bigint	Minimum response time
resp_max	bigint	Maximum response time
resp_avg	bigint	Average response time
resp_total	bigint	Total response time

14.3.85 MPP_TABLES

MPP_TABLES displays information about tables in **PGXC_CLASS**.

Table 14-142 MPP_TABLES columns

Name	Type	Description
schemaname	name	Name of the schema that contains the table
tablename	name	Name of a table
tableowner	name	Owner of the table
tablespace	name	Tablespace where the table is located.
pgroup	name	Name of a node cluster.
nodeoids	oidvector_extend	List of distributed table node OIDs

14.3.86 PG_AVAILABLE_EXTENSION_VERSIONS

PG_AVAILABLE_EXTENSION_VERSIONS displays the extension versions of certain database features.

Table 14-143 PG_AVAILABLE_EXTENSION_VERSIONS columns

Name	Type	Description
name	name	Extension name
version	text	Version name

Name	Type	Description
installed	boolean	The value is **true** if the version of this extension is currently installed.
superuser	boolean	The value is **true** if only system administrators are allowed to install this extension.
relocatable	boolean	The value is **true** if an extension can be relocated to another schema.
schema	name	Name of the schema that the extension must be installed into. The value is null if the extension is partially or fully relocatable.
requires	name[]	Names of prerequisite extensions. The value is null if there are no prerequisite extensions.
comment	text	Comment string from the extension's control file

14.3.87 PG_AVAILABLE_EXTENSIONS

PG_AVAILABLE_EXTENSIONS displays the extended information about certain database features.

Table 14-144 PG_AVAILABLE_EXTENSIONS columns

Name	Type	Description
name	name	Extension name
default_version	text	Name of default version. The value is **NULL** if none is specified.
installed_version	text	Currently installed version of the extension. The value is **NULL** if no version is installed.
comment	text	Comment string from the extension's control file

14.3.88 PG_BULKLOAD_STATISTICS

On any normal node in a cluster, **PG_BULKLOAD_STATISTICS** displays the execution status of the import and export services. Each import or export service corresponds to a record. This view is accessible only to users with system administrators rights.

Table 14-145 PG_BULKLOAD_STATISTICS columns

Name	Type	Description
node_name	text	Node name
db_name	text	Database name
query_id	bigint	Query ID. It is equivalent to **debug_query_id**.
tid	bigint	ID of the current thread
lwtid	integer	Lightweight thread ID
session_id	bigint	GDS session ID
direction	text	Service type. The options are **gds to file**, **gds from file**, **gds to pipe**, **gds from pipe**, **copy from**, and **copy to**.
query	text	Query statement
address	text	Location of the foreign table used for data import and export
query_start	timestamp with time zone	Start time of data import or export
total_bytes	bigint	Total size of data to be processed This parameter is specified only when a GDS common file is to be imported and the record in the row comes from a CN. Otherwise, left this parameter unspecified.
phase	text	Execution phase of the current service import and export. The options are **INITIALIZING**, **TRANSFER_DATA**, and **RELEASE_RESOURCE**.
done_lines	bigint	Number of lines that have been transferred
done_bytes	bigint	Number of bytes that have been transferred

14.3.89 PG_COMM_CLIENT_INFO

PG_COMM_CLIENT_INFO stores the client connection information of a single node. (You can query this view on a DN to view the information about the connection between the CN and DN.)

Table 14-146 PG_COMM_CLIENT_INFO columns

Name	Type	Description
node_name	text	Current node name.
app	text	Client application name
tid	bigint	Thread ID of the current thread.
lwtid	integer	Lightweight thread ID of the current thread.
query_id	bigint	Query ID. It is equivalent to **debug_query_id**.
socket	integer	It is displayed if the connection is a physical connection.
remote_ip	text	Peer node IP address.
remote_port	text	Peer node port.
logic_id	integer	If the connection is a logical connection, **sid** is displayed. If **-1** is displayed, the current connection is a physical connection.

14.3.90 PG_COMM_DELAY

PG_COMM_DELAY displays the communication library delay status for a single DN.

Table 14-147 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
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. Its unit is 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. Its unit is microsecond.

Name	Type	Description
max_delay	integer	Maximum delay of the current physical connection within 1 minute. The unit is microsecond.

14.3.91 PG_COMM_STATUS

PG_COMM_STATUS displays the communication library status for a single DN.

Table 14-148 PG_COMM_STATUS columns

Name	Type	Description
node_name	text	Specifies the node name.
rxpck/s	integer	Receiving rate of the communication library on a node. The unit is byte/s.
txpck/s	integer	Sending rate of the communication library on a node. The unit is byte/s.
rxkB/s	bigint	Receiving rate of the communication library on a node. The unit is KB/s.
txkB/s	bigint	Sending rate of the communication library on a node. The unit is KB/s.
buffer	bigint	Size of the buffer of the Cmailbox.
memKB(libcomm)	bigint	Communication memory size of the **libcomm** process, in KB.
memKB(libpq)	bigint	Communication memory size of the **libpq** process, in KB.
%USED(PM)	integer	Real-time usage of the postmaster thread.
%USED (sflow)	integer	Real-time usage of the **gs_sender_flow_controller** thread.
%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.

14.3.92 PG_COMM_RECV_STREAM

PG_COMM_RECV_STREAM displays the receiving stream status of all the communication libraries for a single DN.

Table 14-149 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	Current status of the stream ● **UNKNOWN**: The logical connection is unknown. ● **READY**: The logical connection is ready. ● **RUN**: The logical connection receives packets normally. ● **HOLD**: The logical connection is waiting to receive packets. ● **CLOSED**: The logical connection is closed. ● **TO_CLOSED**: The logical connection is to 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. The unit is byte.
time	bigint	Current life cycle service duration of the stream. The unit is ms.
speed	bigint	Average receiving rate of the stream. The unit is byte/s.
quota	bigint	Current communication quota value of the stream. The unit is Byte.
buff_usize	bigint	Current size of the data cache of the stream. The unit is byte.

14.3.93 PG_COMM_SEND_STREAM

PG_COMM_SEND_STREAM displays the sending stream status of all the communication libraries for a single DN.

Table 14-150 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	Current status of the stream ● **UNKNOWN**: The logical connection 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 is to 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
send_bytes	bigint	Total data volume sent by the stream. The unit is Byte.
time	bigint	Current life cycle service duration of the stream. The unit is ms.
speed	bigint	Average sending rate of the stream. The unit is Byte/s.

Name	Type	Description
quota	bigint	Current communication quota value of the stream. The unit is Byte.
wait_quota	bigint	Extra time generated when the stream waits the quota value. The unit is ms.

14.3.94 PG_COMM_QUERY_SPEED

PG_COMM_QUERY_SPEED displays traffic information about all queries on a single node.

Table 14-151 PG_COMM_QUERY_SPEED columns

Name	Type	Description
node_name	text	Node name
query_id	bigint	**debug_query_id** corresponding to the stream
rxkB/s	bigint	Receiving rate of the query stream (unit: byte/s)
txkB/s	bigint	Sending rate of the query stream (unit: byte/s)
rxkB	bigint	Total received data of the query stream (unit: byte)
txkB	bigint	Total sent data of the query stream (unit: byte)
rxpck/s	bigint	Packet receiving rate of the query (unit: packets/s)
txpck/s	bigint	Packet sending rate of the query (Unit: packets/s)
rxpck	bigint	Total number of received packets of the query
txpck	bigint	Total number of sent packets of the query

14.3.95 PG_CONTROL_GROUP_CONFIG

PG_CONTROL_GROUP_CONFIG displays the Cgroup configuration information in the system.

Table 14-152 PG_CONTROL_GROUP_CONFIG columns

Name	Type	Description
pg_control_group_config	text	Configuration information of the cgroup

14.3.96 PG_CURSORS

PG_CURSORS displays the cursors that are currently available.

Table 14-153 PG_CURSORS columns

Name	Type	Description
name	text	Cursor name
statement	text	Query statement when the cursor is declared to change
is_holdable	boolean	Whether the cursor is holdable (that is, it can be accessed after the transaction that declared the cursor has committed). If it is, its value is **true**.
is_binary	boolean	Whether the cursor was declared BINARY. If it was, its 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**. Otherwise, the value is **FALSE**.
creation_time	timestamp with time zone	Timestamp at which the cursor is declared

14.3.97 PG_EXT_STATS

PG_EXT_STATS displays extension statistics stored in the **PG_STATISTIC_EXT** table. The extension statistics means multiple columns of statistics.

Table 14-154 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	Name of a table
attname	int2vector	**PG_STATISTIC_EXT**.stakey	Indicates the columns to be combined for collecting statistics.

Name	Type	Reference	Description
inherited	boolean	-	Includes inherited sub-columns if the value is **true**; otherwise, indicates the column in a specified table.
null_frac	real	-	Percentage of column combinations that are null to all records
avg_width	integer	-	Average width of column combinations. The unit is 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 ● The number of distinct values is unknown if the value is **0**. **NOTE** The negated form is used when ANALYZE believes that the number of distinct values is likely to increase as the table grows. 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.
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**.
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_null** will be **NULL**. None of the most common values in **most_common_vals** is **NULL**.

Name	Type	Reference	Description
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_null** 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**)

14.3.98 PG_GET_INVALID_BACKENDS

PG_GET_INVALID_BACKENDS displays the information about backend threads on the CN that are connected to the current standby DN.

Table 14-155 PG_GET_INVALID_BACKENDS columns

Name	Type	Description
pid	bigint	Thread ID
node_name	text	Node information connected to the backend thread
dbname	name	Name of the connected database
backend_start	timestamp with time zone	Backend thread startup time
query	text	Query statement performed by the backend thread

14.3.99 PG_GET_SENDERS_CATCHUP_TIME

PG_GET_SENDERS_CATCHUP_TIME displays the catchup information of the currently active primary/standby instance sending thread on a single DN.

Table 14-156 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
catchup_type	text	Catchup task type, full or incremental
catchup_bcm_filename	text	BCM file executed by the current catchup task
catchup_bcm_finished	integer	Number of BCM files completed by a catchup task
catchup_bcm_total	integer	Total number of BCM files to be operated by a catchup task
catchup_percent	text	Completion percentage of a catchup task
catchup_remaining_time	text	Estimated remaining time of a catchup task

14.3.100 PG_GROUP

PG_GROUP displays the database role authentication and the relationship between roles.

Table 14-157 PG_GROUP columns

Name	Type	Description
groname	name	Group name
grosysid	oid	Group ID
grolist	oid[]	An array, including all the role IDs in this group

14.3.101 PG_INDEXES

PG_INDEXES displays access to useful information about each index in the database.

Table 14-158 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**.spcname	Name of the tablespace that contains the index
indexdef	text	-	Index definition (a reconstructed **CREATE INDEX** command)

Example

Query the index information about a specified table.

```
SELECT * FROM pg_indexes WHERE tablename = 'mytable';
 schemaname | tablename |   indexname    | tablespace |                              indexdef
------------+-----------+----------------+------------
+-------------------------------------------------------------------------------
 public     | mytable   | idx_mytable_id |            | CREATE INDEX idx_mytable_id ON mytable USING btree
(id) TABLESPACE pg_default
(1 row)
```

Query information about indexes of all tables in a specified schema in the current database.

```
SELECT tablename, indexname, indexdef FROM pg_indexes WHERE schemaname = 'public' ORDER BY
tablename,indexname;
 tablename |     indexname      |                                          indexdef
-----------+--------------------
+----------------------------------------------------------------------------------------------------
 books     | books_pkey         | CREATE UNIQUE INDEX books_pkey ON books USING btree (id) TABLESPACE
pg_default
 books     | idx_books_tags_gin | CREATE INDEX idx_books_tags_gin ON books USING gin (tags)
TABLESPACE pg_default
 customer  | c_custkey_key      | CREATE UNIQUE INDEX c_custkey_key ON customer USING btree
(c_custkey, c_name) TABLESPACE pg_default
 mytable   | idx_mytable_id     | CREATE INDEX idx_mytable_id ON mytable USING btree (id) TABLESPACE
pg_default
 test1     | idx_test_id        | CREATE INDEX idx_test_id ON test1 USING btree (id) TABLESPACE pg_default
 v0        | v0_pkey            | CREATE UNIQUE INDEX v0_pkey ON v0 USING btree (c) TABLESPACE pg_default
(6 rows)
```

14.3.102 PG_JOB

PG_JOB displays detailed information about scheduled tasks created by users.

The **PG_JOB** view replaces the **PG_JOB** system catalog in earlier versions and provides forward compatibility with earlier versions. The original **PG_JOB** system catalog is changed to the **PG_JOBS** system catalog. For details about **PG_JOBS**, see **PG_JOBS**.

Table 14-159 PG_JOB columns

Name	Type	Description
job_id	bigint	Job ID
current_postgres_pid	bigint	If the current job has been executed, the PostgreSQL thread ID of this job is recorded. The default value is **-1**, indicating that the task is not executed or has been executed.
log_user	name	User name of the job creator
priv_user	name	User name of the job executor
dbname	name	Name of the database where the job is executed
node_name	name	CN node on which the job will be created and executed
job_status	text	Status of the current job. The value range is **r**, **s**, **f**, or **d**. The default value is **s**. The indications are as follows: ● r=running ● s=successfully finished ● f=job failed ● d=disable **NOTE** ● Note: When you disable a scheduled task (by setting **job_queue_processes** to **0**), the thread monitor the job execution is not started, and the **job_status** will not be updated. You can ignore the **job_status**. ● Only when the scheduled task function is enabled (that is, when **job_queue_processes** is not **0**), the system updates the value of **job_status** based on the real-time job status.
start_date	timestamp without time zone	Start time of the first job execution, precise to millisecond
next_run_date	timestamp without time zone	Scheduled time of the next job execution, accurate to millisecond
failure_count	smallint	Number of consecutive failures.
interval	text	Job execution interval

Name	Type	Description
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
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 namespace where a job is running
what	text	Job content

14.3.103 PG_JOB_PROC

The **PG_JOB_PROC** view replaces the **PG_JOB_PROC** system catalog in earlier versions and provides forward compatibility with earlier versions. The original **PG_JOB_PROC** and **PG_JOB** system catalogs are merged into the **PG_JOBS** system catalog in the current version. For details about the **PG_JOBS** system catalog, see **PG_JOBS**.

Table 14-160 PG_JOB_PROC columns

Name	Type	Description
job_id	bigint	Job ID
what	text	Job content

14.3.104 PG_JOB_SINGLE

PG_JOB_SINGLE displays job information about the current node.

Table 14-161 PG_JOB_SINGLE columns

Name	Type	Description
job_id	bigint	Job ID

Name	Type	Description
current_postgres_pid	bigint	If the current job has been executed, the PostgreSQL thread ID of this job is recorded. The default value is **-1**, indicating that the task is not executed or has been executed.
log_user	name	User name of the job creator
priv_user	name	User name of the job executor
dbname	name	Name of the database where the job is executed
node_name	name	CN node on which the job will be created and executed
job_status	text	Status of the current job. The value range is **r**, **s**, **f**, or **d**. The default value is **s**. The indications are as follows: ● r=running ● s=successfully finished ● f=job failed ● d=disable **NOTE** ● Note: When you disable a scheduled task (by setting **job_queue_processes** to **0**), the thread monitor the job execution is not started, and the **job_status** will not be updated. You can ignore the **job_status**. ● Only when the scheduled task function is enabled (that is, when **job_queue_processes** is not **0**), the system updates the value of **job_status** based on the real-time job status.
start_date	timestamp without time zone	Start time of the first job execution, precise to millisecond
next_run_date	timestamp without time zone	Scheduled time of the next job execution, accurate to millisecond
failure_count	smallint	Number of consecutive failures.
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 namespace where a job is running
what	text	Job content

14.3.105 PG_LIFECYCLE_DATA_DISTRIBUTE

PG_LIFECYCLE_DATA_DISTRIBUTE displays the distribution of cold and hot data in a multi-temperature table of OBS.

Table 14-162 PG_LIFECYCLE_DATA_DISTRIBUTE columns

Name	Type	Description
schemaname	name	Schema name
tablename	name	Current table name
nodename	name	Node name
hotpartition	text	Hot partition on the DN
coldpartition	text	Cold partition on the DN
switchablepartition	text	Switchable partition on the DN
hotdatasize	text	Data size of the hot partition on the DN
colddatasize	text	Data size of the cold partition on the DN
switchabledatasize	text	Data size of the switchable partition on the DN

14.3.106 PG_LOCKS

PG_LOCKS displays information about the locks held by open transactions.

Table 14-163 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	**PG_DATABASE**.oid	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	**PG_CLASS**.oid	OID of the relationship targeted by the lock. The value is **NULL** if the object is neither a relationship nor part of a relationship.
page	integer	-	Page number targeted by the lock within the relationship. If the object is neither a relation page nor row page, the value is **NULL**.
tuple	smallint	-	Row number targeted by the lock within the page. If the object is not a row, the value is **NULL**.
virtualxid	text	-	Virtual ID of the transaction targeted by the lock. If the object is not a virtual transaction ID, the value is **NULL**.
transactionid	xid	-	ID of the transaction targeted by the lock. If the object is not a transaction ID, the value is **NULL**.
classid	oid	**PG_CLASS**.oid	OID of the system table that contains the object. If the object is not a general database object, the value is **NULL**.
objid	oid	-	OID of the lock target within its system table. If the target is not a general database object, the value is **NULL**.
objsubid	smallint	-	Column number for a column in the table. The value is **0** if the target is some other object type. If the object is not a general database object, the value is **NULL**.

Name	Type	Reference	Description
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. This is **NULL** if the lock is held by a prepared transaction.
mode	text	-	Lock mode held or desired by this thread For more information about lock modes, see **LOCK**.
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	-	Whether the lock is obtained through **fast-path** (**true**) or main lock table (**false**)

14.3.107 PG_NODE_ENV

PG_NODE_ENVO displays the environmental variable information about the current node.

Table 14-164 PG_NODE_ENV columns

Name	Type	Description
node_name	text	Name of the current node
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 current node
datapath	text	Data directory of the node
log_directory	text	Log directory of the node

14.3.108 PG_OS_THREADS

PG_OS_THREADS displays the status information about all the threads under the current node.

Table 14-165 PG_OS_THREADS columns

Name	Type	Description
node_name	text	Name of the current node
pid	bigint	Thread number running under the current node process
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

14.3.109 PG_POOLER_STATUS

PG_POOLER_STATUS displays the cache connection status in the pooler. **PG_POOLER_STATUS** can only query on the CN, and displays the connection cache information about the pooler module.

Table 14-166 PG_POOLER_STATUS columns

Name	Type	Description
database	text	Database name
user_name	text	Username
tid	bigint	ID of a thread connected to the CN
node_oid	bigint	OID of the node connected
node_name	name	Name of the node connected
in_use	boolean	Whether the connection is in use ● **t** (true): indicates that the connection is in use. ● **f** (false): indicates that the connection is not in use.
fdsock	bigint	Peer socket
remote_pid	bigint	Peer thread ID
session_params	text	GUC session parameter delivered by the connection.

Example

View information about the connection pool **pooler**:

```
select database,user_name,node_name,in_use,count(*) from pg_pooler_status group by 1, 2, 3 ,4 order by 5
desc limit 50;
 database | user_name |  node_name   | in_use | count
----------+-----------+--------------+--------+-------
 mydbdemo | user3     | cn_5001      | f      |     2
 mydbdemo | user3     | dn_6005_6006 | t      |     2
 mydbdemo | user3     | dn_6001_6002 | t      |     2
 mydbdemo | user3     | dn_6003_6004 | f      |     2
 mydbdemo | user3     | dn_6003_6004 | t      |     2
 mydbdemo | user3     | dn_6005_6006 | f      |     2
 mydbdemo | user3     | dn_6001_6002 | f      |     2
 mydbdemo | user3     | cn_5002      | f      |     2
 gaussdb  | user3     | dn_6003_6004 | f      |     1
 mydbdemo | user3     | cn_5001      | t      |     1
 music    | user2     | dn_6003_6004 | f      |     1
 music    | user2     | dn_6005_6006 | f      |     1
 gaussdb  | user1     | dn_6005_6006 | f      |     1
(13 rows)
```

14.3.110 PG_PREPARED_STATEMENTS

PG_PREPARED_STATEMENTS displays all prepared statements that are available in the current session.

Table 14-167 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 casting the regtype value to oid.
from_sql	boolean	How a prepared statement was created ● **true**: The prepared statement was created through the PREPARE statement. ● **false** The statement was prepared through the frontend/backend protocol.

14.3.111 PG_PREPARED_XACTS

PG_PREPARED_XACTS displays information about transactions that are currently prepared for two-phase commit.

Table 14-168 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

14.3.112 PG_QUERYBAND_ACTION

PG_QUERYBAND_ACTION displays information about the object associated with **query_band** and the **query_band** query order.

Table 14-169 PG_QUERYBAND_ACTION columns

Name	Type	Description
qband	text	**query_band** key-value pairs
respool_id	oid	OID of the resource pool associated with **query_band**
respool	text	Name of the resource pool associated with **query_band**
priority	text	Intra-queue priority associated with **query_band**
qborder	integer	**query_band** query order

14.3.113 PG_REPLICATION_SLOTS

PG_REPLICATION_SLOTS displays the replication node information.

Table 14-170 PG_REPLICATION_SLOTS columns

Name	Type	Description
slot_name	text	Name of a replication node
plugin	name	Name of the output plug-in of the logical replication slot
slot_type	text	Type of a replication node
datoid	oid	OID of the database on the replication node
database	name	Name of the database on the replication node
active	boolean	Whether the replication node is active
xmin	xid	Transaction ID of the replication node
catalog_xmin	text	ID of the earliest-decoded transaction corresponding to the logical replication slot
restart_lsn	text	Xlog file information on the replication node
dummy_standby	boolean	Whether the replication node is the dummy standby node

14.3.114 PG_ROLES

PG_ROLES displays information about database roles.

Table 14-171 PG_ROLES columns

Name	Type	Reference	Description
rolname	name	-	Role name
rolsuper	boolean	-	Whether the role is the initial system administrator with the highest permission
rolinherit	boolean	-	Whether the role inherits permissions for this type of roles
rolcreaterole	boolean	-	Whether the role can create other roles
rolcreatedb	boolean	-	Whether the role can create databases

Name	Type	Reference	Description
rolcatupdate	boolean	-	Whether the role can update system tables directly. Only the initial system administrator whose usesysid is 10 has this permission. It is not available 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	-	Limits the maximum number of concurrent connections of a user on a CN node. -1 indicates no limit.
rolpassword	text	-	Not the password (always reads as ********)
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
rolparentid	oid	**PG_AUTHID**.rolparentid	OID of a group user to which the user belongs
roltabspace	text	-	The storage space of the user permanent table.
roltempspace	text	-	The storage space of the user temporary table.
rolspillspace	text	-	The operator disk flushing space of the user.
rolconfig	text[]	-	Session defaults for runtime configuration variables
oid	oid	**PG_AUTHID**.oid	ID of the role
roluseft	boolean	**PG_AUTHID**.roluseft	Whether the role can perform operations on foreign tables

Name	Type	Reference	Description
nodegroup	name	-	Name of the logical cluster associated with the role. If no logical cluster is associated, this column is left empty.

14.3.115 PG_RULES

PG_RULES displays information about rewrite rules.

Table 14-172 PG_RULES columns

Name	Type	Description
schemaname	name	Name of the schema that contains the table
tablename	name	Name of the table the rule is for
rulename	name	Rule name
definition	text	Rule definition (a reconstructed creation command)

14.3.116 PG_RUNNING_XACTS

PG_RUNNING_XACTS displays information about running transactions on the current node.

Table 14-173 PG_RUNNING_XACTS columns

Name	Type	Description
handle	integer	Handle corresponding to the transaction in GTM
gxid	xid	Transaction ID
state	tinyint	Transaction status (**3**: prepared or **0**: starting)
node	text	Node name
xmin	xid	Minimum transaction ID **xmin** 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** status. If the status is not **prepared**, the value is **0**.

Name	Type	Description
pid	bigint	Thread ID corresponding to the transaction
next_xid	xid	Transaction ID sent from a CN to a DN

14.3.117 PG_SECLABELS

PG_SECLABELS displays information about security labels.

Table 14-174 PG_SECLABEL columns

Name	Type	Reference	Description
objoid	oid	Any OID column	OID of the object this security label pertains to
classoid	oid	**PG_CLASS**.oid	OID of the system table that contains the object
objsubid	integer	-	For a security label on a table column, this is the column number (the **objoid** and **classoid** refer to the table itself). For all other object types, this column is **0**.
objtype	text	-	Type of the object to which this label applies
objnamespace	oid	**PG_NAMESPACE**.oid	OID of the namespace for this object, if applicable; otherwise NULL.
objname	text	-	Name of the object to which the label applies
provider	text	**PG_SECLABEL**.provider	Label provider associated with this label
label	text	**PG_SECLABEL**.label	Security label applied to this object

14.3.118 PG_SESSION_WLMSTAT

PG_SESSION_WLMSTAT displays the corresponding load management information about the task currently executed by the user.

Table 14-175 PG_SESSION_WLMSTAT columns

Column	Type	Description
datid	oid	OID of the database this backend is connected to

Column	Type	Description
datname	name	Name of the database the backend is connected to
threadid	bigint	ID of the backend thread
processid	integer	PID of the backend thread
usesysid	oid	OID of the user who logged into the backend
appname	text	Name of the application that is 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	Statement attributes ● **Ordinary**: default attribute of a statement before it is parsed by the database ● **Simple**: simple statements ● **Complicated**: complicated statements ● **Internal**: internal statement of the database
block_time	bigint	Pending duration of the statements by now (unit: s)
elapsed_time	bigint	Actual execution duration of the statements by now (unit: s)
total_cpu_time	bigint	Total CPU usage duration of the statement on the DN in the last period (unit: s)
cpu_skew_percent	integer	CPU usage inclination ratio of the statement on the DN in the last period
statement_mem	integer	Estimated memory required for statement execution.
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	Cgroup currently used by the statement

Column	Type	Description
status	text	Status of a statement, including: ● **pending** ● **running** ● **finished** (If **enqueue** is set to **StoredProc** or **Transaction**, this state indicates that only some of the 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	Current queuing status of the statements, including: ● **Global**: global queuing. ● **Respool**: resource pool queuing. ● **CentralQueue**: queuing on the CCN ● **Transaction**: being in a transaction block ● **StoredProc**: being in a stored procedure ● **None**: not in a 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	Text of this backend's most recent query If **state** is **active**, this column shows the executing query. In all other states, it shows the last query that was executed.
isplana	bool	In logical cluster mode, indicates whether a statement occupies the resources of other logical clusters. The default value is **f**, indicating that resources of other logical clusters are not occupied.
node_group	text	Logical cluster of the user running the statement
lane	text	Fast or slow lane for statement queries. ● fast: fast lane ● slow: slow lane ● none: not controlled

14.3.119 PG_SESSION_IOSTAT

PG_SESSION_IOSTAT has been discarded in version 8.1.2 and is reserved for compatibility with earlier versions. This view is invalid in the current version.

Table 14-176 PG_SESSION_IOSTAT columns

Name	Type	Description
query_id	bigint	Job ID
mincurriops	integer	Minimum I/O of the current job across DNs
maxcurriops	integer	Maximum I/O of the current job across DNs
minpeakiops	integer	Minimum peak I/O of the current job across DNs
maxpeakiops	integer	Maximum peak I/O of the current job across DNs
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 cluster of the user running the job

14.3.120 PG_SETTINGS

PG_SETTINGS displays information about parameters of the running database.

Table 14-177 PG_SETTINGS columns

Name	Type	Description
name	text	Parameter name
setting	text	Current value of the parameter
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 of parameter values including internal, postmaster, sighup, backend, superuser, and user
vartype	text	Parameter type. It can be **bool**, **enum**, **integer**, **real**, or **string**.
source	text	Method of assigning the parameter value

Name	Type	Description
min_val	text	Minimum value of the parameter. If the parameter type is not numeric data, the value of this column is null.
max_val	text	Maximum value of the parameter. If the parameter type is not numeric data, the value of this column is null.
enumvals	text[]	Valid values of an enum-typed 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.

14.3.121 PG_SHADOW

PG_SHADOW displays properties of all roles that are marked as **rolcanlogin** in **PG_AUTHID**.

This view is not readable to all users because it contains passwords. **PG_USER** is a publicly readable view on **PG_SHADOW** that blanks out the password column.

Table 14-178 PG_SHADOW columns

Name	Type	Reference	Description
usename	name	**PG_AUTHID**.rolname	User name
usesysid	oid	**PG_AUTHID**.oid	ID of a user
usecreatedb	boolean	-	Indicates that the user can create databases.
usesuper	boolean	-	Indicates that the user is an administrator.

Name	Type	Reference	Description
usecatupd	boolean	-	Indicates that the user can update system catalogs. Even the system administrator cannot do this unless this column is **true**.
userepl	boolean	-	User can initiate streaming replication and put the system in and out of backup mode.
passwd	text	-	Password (possibly encrypted); null if none. See **PG_AUTHID** for details about how encrypted passwords are stored.
valbegin	timestamp with time zone	-	Account validity start time; null if no start time
valuntil	timestamp with time zone	-	Password expiry time; null if no expiration
respool	name	-	Resource pool used by the user
parent	oid	-	Parent resource pool
spacelimit	text	-	The storage space of the permanent table.
tempspacelimit	text	-	The storage space of the temporary table.
spillspacelimit	text	-	The operator disk flushing space.
useconfig	text[ ]	-	Session defaults for runtime configuration variables

14.3.122 PG_SHARED_MEMORY_DETAIL

PG_SHARED_MEMORY_DETAIL displays usage information about all the shared memory contexts.

Table 14-179 PG_SHARED_MEMORY_DETAIL columns

Name	Type	Description
contextname	text	Name of the context in the memory
level	smallint	Hierarchy of the memory context
parent	text	Context of the parent memory
totalsize	bigint	Total size of the shared memory, in bytes.
freesize	bigint	Remaining size of the shared memory, in bytes.
usedsize	bigint	Used size of the shared memory, in bytes.

14.3.123 PG_STATS

PG_STATS displays the single-column statistics stored in the **pg_statistic** table.

Table 14-180 PG_STATS columns

Name	Type	Reference	Description
schemaname	name	**PG_NAMESPACE**.nspname	Name of the schema that contains the table
tablename	name	**PG_CLASS**.relname	Name of the table
attname	name	**PG_ATTRIBUTE**.attname	Column name
inherited	boolean	-	Includes inherited sub-columns if the value is **true**; otherwise, 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 The negated form is used when ANALYZE believes that the number of distinct values is likely to increase as the table grows. The positive form is used when the column seems to have a fixed number of possible values. For example, **-1** indicates a unique column in which the number of distinct values is the same as the number of rows.
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. If this combination does not have the most common values, it will be **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**)

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 field is null if the field 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	-	Specifies a list of non-null element values most often appearing.
most_common_elem_freqs	real[]	-	Specifies a list of the frequencies of the most common element values.
elem_count_histogram	real[]	-	Specifies a histogram of the counts of distinct non-null element values.

14.3.124 PG_STAT_ACTIVITY

PG_STAT_ACTIVITY displays information about the current user's queries. If you have the rights of an administrator or the preset role, you can view all information about user queries.

Table 14-181 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	Backend thread ID
lwtid	integer	Lightweight thread ID
usesysid	oid	OID of the user logging in to the backend
usename	name	OID of the user logging 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 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 only be non-null 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, or **-1** if a Unix socket is used
backend_start	timestamp with time zone	Startup time of the backend process, that is, the time when the client connects to the server.
xact_start	timestamp with time zone	Time when the current transaction was started, or **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 if **state** is not **active**, when the last query was started
state_change	timestamp with time zone	Time for the last status change
waiting	boolean	The value is **t** if the backend is currently waiting for a lock or node. Otherwise, the value is **f**.

Name	Type	Description
enqueue	text	Queuing status of a statement. Its value can be: ● **waiting in global queue**: The statement is queuing in the global concurrent queue. The number of concurrent statements exceeds the value of **max_active_statements** configured for a single CN. ● **waiting in respool queue**: The statement is queuing in the resource pool and the concurrency of simple jobs is limited. The main reason is that the concurrency of simple jobs exceeds the upper limit **max_dop** of the fast track. ● **waiting in ccn queue**: The job is in the CCN queue, which may be global memory queuing, slow lane memory queuing, or concurrent queuing. The scenarios are: – The available global memory exceeds the upper limit, the job is queuing in the global memory queue. – Concurrent requests on the slow lane in the resource pool exceed the upper limit, which is specified by **active_statements**. – The slow lane memory of the resource pool exceeds the upper limit, that is, the estimated memory of concurrent jobs in the resource pool exceeds the upper limit specified by **mem_percent**. ● Empty or **no waiting queue**: The statement is running.

Name	Type	Description
state	text	Current overall state of this backend. Its value can be: ● **active**: The backend is executing queries. ● **idle**: The backend is waiting for new client commands. ● **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 only their own session status. The state information of other accounts is empty.
resource_pool	name	Resource pool used by the user
stmt_type	text	Statement type
query_id	bigint	ID of a query
query	text	Text of the most recent query in this backend If **state** is **active**, this column shows the running 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**)

14.3.125 PG_STAT_ALL_INDEXES

PG_STAT_ALL_INDEXES displays statistics about all accesses to a specific index in the current database.

Indexes can be used via either simple index scans or "bitmap" index scans. Bitmap scans can combine the output of multiple indexes using AND or OR rules, but combining independent row fetching with specific indexes is challenging. Consequently, a bitmap scan increases the index count in **pg_stat_all_indexes.idx_tup_read** and the table count in **pg_stat_all_tables.idx_tup_fetch**, while having no effect on **pg_stat_all_indexes.idx_tup_fetch**.

Table 14-182 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 this index is in
relname	name	Name of the table for this index
indexrelname	name	Name of this index
idx_scan	bigint	Number of index scans initiated on this index
idx_tup_read	bigint	Number of index entries returned by scans on this index
idx_tup_fetch	bigint	Number of live table rows fetched by simple index scans using this index

14.3.126 PG_STAT_ALL_TABLES

PG_STAT_ALL_TABLES displays statistics about accesses to tables in the current database, including TOAST tables.

Table 14-183 PG_STAT_ALL_TABLES columns

Name	Type	Description
relid	oid	Table OID
schemaname	name	Schema name of the table
relname	name	Name of the table
seq_scan	bigint	Number of sequential scans started on the table
seq_tup_read	bigint	Number of rows that have live data fetched by sequential scans
idx_scan	bigint	Number of index scans
idx_tup_fetch	bigint	Number of rows that have live data fetched by index scans
n_tup_ins	bigint	Number of rows inserted

Name	Type	Description
n_tup_upd	bigint	Number of rows updated
n_tup_del	bigint	Number of rows deleted
n_tup_hot_upd	bigint	Number of rows updated by **HOT** (no separate index update is 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 (excluding **VACUUM FULL**)
last_autovacuum	timestamp with time zone	Last time at which this table was automatically vacuumed
last_analyze	timestamp with time zone	Last time at which this table was analyzed
last_autoanalyze	timestamp with time zone	Last time at which this table was automatically vacuumed
vacuum_count	bigint	Number of vacuum operations (excluding **VACUUM FULL**)
autovacuum_count	bigint	Number of autovacuum operations
analyze_count	bigint	Number of analyze operations
autoanalyze_count	bigint	Number of autoanalyze operations
last_data_changed	timestamp with time zone	Last time at which this table was updated (by **INSERT**/**UPDATE**/**DELETE** or **EXCHANGE**/**TRUNCATE**/**DROP** *partition*). This column is recorded only on the local CN.

Example

Query the last data change time in the **table_test** table:

```
SELECT last_data_changed FROM PG_STAT_ALL_TABLES WHERE relname ='table_test';
       last_data_changed
-------------------------------
 2024-03-27 10:28:16.277136+08
(1 row)
```

14.3.127 PG_STAT_BAD_BLOCK

PG_STAT_BAD_BLOCK displays statistics about page or CU verification failures after a node is started.

Table 14-184 PG_STAT_BAD_BLOCK columns

Name	Type	Description
nodename	text	Node name
databaseid	integer	Database OID
tablespaceid	integer	Tablespace OID
relfilenode	integer	File object ID
forknum	integer	File type
error_count	integer	Number of verification failures
first_time	timestamp with time zone	Time of the first occurrence
last_time	timestamp with time zone	Time of the latest occurrence

14.3.128 PG_STAT_BGWRITER

PG_STAT_BGWRITER displays statistics about the background writer process's activity.

Table 14-185 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	Time spent writing files to disks during checkpoints, in milliseconds.
checkpoint_sync_time	double precision	Time spent in synchronizing data to disks during checkpoints, in milliseconds.
buffers_checkpoint	bigint	Number of buffers written during checkpoints

Name	Type	Description
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 that a backend has to execute **fsync**
buffers_alloc	bigint	Number of buffers allocated
stats_reset	timestamp with time zone	Time at which these statistics were reset

14.3.129 PG_STAT_DATABASE

PG_STAT_DATABASE displays the status and statistics of each database on the current node.

Table 14-186 PG_STAT_DATABASE columns

Name	Type	Description
datid	oid	Database OID
datname	name	Database name
numbackends	integer	Number of backends currently connected to this database on the current node. This is the only column in this view that reflects the current state value. All columns return the accumulated value since the last reset.
xact_commit	bigint	Number of transactions in this database that have been committed on the current node
xact_rollback	bigint	Number of transactions in this database that have been rolled back on the current node
blks_read	bigint	Number of disk blocks read in this database on the current node
blks_hit	bigint	Number of disk blocks found in the buffer cache on the current node, that is, the number of blocks hit in the cache. (This only includes hits in the GaussDB(DWS) buffer cache, not in the file system cache.)

Name	Type	Description
tup_returned	bigint	Number of rows returned by queries in this database on the current node
tup_fetched	bigint	Number of rows fetched by queries in this database on the current node
tup_inserted	bigint	Number of rows inserted in this database on the current node
tup_updated	bigint	Number of rows updated in this database on the current node
tup_deleted	bigint	Number of rows deleted from this database on the current node
conflicts	bigint	Number of queries canceled due to database recovery conflicts on the current node (conflicts occurring only on the standby server). For details, see **PG_STAT_DATABASE_CONFLICTS**.
temp_files	bigint	Number of temporary files created by this database on the current node. 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	Size of temporary files written to this database on the current node. 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 in this database on the current node
blk_read_time	double precision	Time spent reading data file blocks by backends in this database on the current node, in milliseconds
blk_write_time	double precision	Time spent writing into data file blocks by backends in this database on the current node, in milliseconds
stats_reset	timestamp with time zone	Time when the database statistics are reset on the current node

14.3.130 PG_STAT_DATABASE_CONFLICTS

PG_STAT_DATABASE_CONFLICTS displays statistics about database conflicts.

Table 14-187 PG_STAT_DATABASE_CONFLICTS columns

Name	Type	Description
datid	oid	Database OID
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
confl_deadlock	bigint	Number of conflicting deadlocks

14.3.131 PG_STAT_GET_MEM_MBYTES_RESERVED

PG_STAT_GET_MEM_MBYTES_RESERVED displays the current activity information of a thread stored in memory. You need to specify the thread ID (pid in **PG_STAT_ACTIVITY**) for query. If the thread ID is set to **0**, the current thread ID is used. For example:

```
SELECT pg_stat_get_mem_mbytes_reserved(0);
```

Table 14-188 PG_STAT_GET_MEM_MBYTES_RESERVED columns

Parameter	Description
ConnectInfo	Connection information
ParctlManager	Concurrency management information
GeneralParams	Basic parameter information
GeneralParams RPDATA	Basic resource pool information
ExceptionManager	Exception management information
CollectInfo	Collection information
GeneralInfo	Basic information
ParctlState	Concurrency status information
CPU INFO	CPU information
ControlGroup	Cgroup information
IOSTATE	I/O status information

14.3.132 PG_STAT_USER_FUNCTIONS

PG_STAT_USER_FUNCTIONS displays user-defined function status information in the namespace. (The language of the function is non-internal language.)

Table 14-189 PG_STAT_USER_FUNCTIONS columns

Name	Type	Description
funcid	oid	Function OID
schemaname	name	Schema name
funcname	name	Name of the function
calls	bigint	Number of times this 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 itself, excluding other functions called by it

14.3.133 PG_STAT_USER_INDEXES

PG_STAT_USER_INDEXES displays information about the index status of user-defined ordinary tables and TOAST tables.

Table 14-190 PG_STAT_USER_INDEXES columns

Name	Type	Description
relid	oid	Table OID for the index
indexrelid	oid	OID of this index
schemaname	name	Name of the schema this index is in
relname	name	Name of the table for this index
indexrelname	name	Name of this index
idx_scan	bigint	Number of index scans
idx_tup_read	bigint	Number of index entries returned by scans on this index
idx_tup_fetch	bigint	Number of rows that have live data fetched by index scans

14.3.134 PG_STAT_USER_TABLES

PG_STAT_USER_TABLES displays status information about user-defined ordinary tables and TOAST tables in all namespaces.

Table 14-191 PG_STAT_USER_TABLES columns

Name	Type	Description
relid	oid	Table OID
schemaname	name	Schema name of the table
relname	name	Name of a table
seq_scan	bigint	Number of sequential scans started on the table
seq_tup_read	bigint	Number of rows that have live data fetched by sequential scans
idx_scan	bigint	Number of index scans
idx_tup_fetch	bigint	Number of rows that have live data 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 updated by **HOT** (no separate index update is 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 (excluding **VACUUM FULL**)
last_autovacuum	timestamp with time zone	Last time at which this table was automatically vacuumed
last_analyze	timestamp with time zone	Last time at which this table was analyzed
last_autoanalyze	timestamp with time zone	Time of the last **AUTOANALYZE**
vacuum_count	bigint	Number of vacuum operations (excluding **VACUUM FULL**)

Name	Type	Description
autovacuum_count	bigint	Number of autovacuum operations
analyze_count	bigint	Number of analyze operations
autoanalyze_count	bigint	Number of autoanalyze operations

14.3.135 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 14-192 PG_STAT_REPLICATION columns

Name	Type	Description
pid	bigint	PID of the thread
usesysid	oid	User system ID
usename	name	Username
application_name	text	Application name
client_addr	inet	Client address.
client_hostname	text	Client name
client_port	integer	Client port number
backend_start	timestamp with time zone	Start time of the program
state	text	Log replication state (catch-up or consistent streaming)
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 duplication, synchronous duplication, or potential synchronization)

14.3.136 PG_STAT_SYS_INDEXES

PG_STAT_SYS_INDEXES displays the index status information about all the system catalogs in the **pg_catalog** and **information_schema** schemas.

Table 14-193 PG_STAT_SYS_INDEXES columns

Name	Type	Description
relid	oid	Table OID for the index
indexrelid	oid	OID of this index
schemaname	name	Name of the schema this index is in
relname	name	Name of the table for this index
indexrelname	name	Name of this index
idx_scan	bigint	Number of index scans
idx_tup_read	bigint	Number of index entries returned by scans on this index
idx_tup_fetch	bigint	Number of rows that have live data fetched by index scans

14.3.137 PG_STAT_SYS_TABLES

PG_STAT_SYS_TABLES displays the statistics about the system catalogs of all the namespaces in **pg_catalog** and **information_schema** schemas.

Table 14-194 PG_STAT_SYS_TABLES columns

Name	Type	Description
relid	oid	Table OID
schemaname	name	Schema name of the table
relname	name	Name of a table
seq_scan	bigint	Number of sequential scans started on the table

Name	Type	Description
seq_tup_read	bigint	Number of rows that have live data fetched by sequential scans
idx_scan	bigint	Number of index scans
idx_tup_fetch	bigint	Number of rows that have live data 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 updated by **HOT** (no separate index update is 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 (excluding **VACUUM FULL**)
last_autovacuum	timestamp with time zone	Last time at which this table was automatically vacuumed
last_analyze	timestamp with time zone	Last time at which this table was analyzed
last_autoanalyze	timestamp with time zone	Last time at which this table was automatically analyzed
vacuum_count	bigint	Number of vacuum operations (excluding **VACUUM FULL**)
autovacuum_count	bigint	Number of autovacuum operations
analyze_count	bigint	Number of analyze operations
autoanalyze_count	bigint	Number of autoanalyze operations

14.3.138 PG_STAT_XACT_ALL_TABLES

PG_STAT_XACT_ALL_TABLES displays the transaction status information about all ordinary tables and TOAST tables in the namespaces.

Table 14-195 PG_STAT_XACT_ALL_TABLES columns

Name	Type	Description
relid	oid	Table OID
schemaname	name	Schema name of the table
relname	name	Name of a table
seq_scan	bigint	Number of sequential scans started on the table
seq_tup_read	bigint	Number of live rows fetched by sequential scans
idx_scan	bigint	Number of index scans started 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 with HOT updates (no separate index update is required).

14.3.139 PG_STAT_XACT_SYS_TABLES

PG_STAT_XACT_SYS_TABLES displays the transaction status information of the system catalog in the namespace.

Table 14-196 PG_STAT_XACT_SYS_TABLES columns

Name	Type	Description
relid	oid	Table OID
schemaname	name	Schema name of the table
relname	name	Table name
seq_scan	bigint	Number of sequential scans started on the table
seq_tup_read	bigint	Number of live rows fetched by sequential scans
idx_scan	bigint	Number of index scans started 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_up d	bigint	Number of rows with HOT updates (no separate index update is required).

14.3.140 PG_STAT_XACT_USER_FUNCTIONS

PG_STAT_XACT_USER_FUNCTIONS displays statistics about function execution.

Table 14-197 PG_STAT_XACT_USER_FUNCTIONS columns

Name	Type	Description
funcid	oid	Function OID
schemaname	name	Schema name
funcname	name	Name of the function
calls	bigint	Number of times this 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 itself, excluding other functions called by it

14.3.141 PG_STAT_XACT_USER_TABLES

PG_STAT_XACT_USER_TABLES displays the transaction status information of the user table in the namespace.

Table 14-198 PG_STAT_XACT_USER_TABLES columns

Name	Type	Description
relid	oid	Table OID
schemaname	name	Schema name of the table
relname	name	Name of a table
seq_scan	bigint	Number of sequential scans started on the table
seq_tup_read	bigint	Number of live rows fetched by sequential scans
idx_scan	bigint	Number of index scans started on the table

Name	Type	Description
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 with HOT updates (no separate index update is required).

14.3.142 PG_STATIO_ALL_INDEXES

PG_STATIO_ALL_INDEXES displays I/O statistics of all indexes in the current database.

Table 14-199 PG_STATIO_ALL_INDEXES columns

Name	Type	Description
relid	oid	OID of the index table
indexrelid	oid	OID of this index
schemaname	name	Name of the schema this index is in
relname	name	Name of the table for this index
indexrelname	name	Name of this index
idx_blks_read	bigint	Number of disk blocks read from the index
idx_blks_hit	bigint	Number of buffer hits in this index

14.3.143 PG_STATIO_ALL_SEQUENCES

PG_STATIO_ALL_SEQUENCES displays the sequence information in the current database and the I/O statistics of a specified sequence.

Table 14-200 PG_STATIO_ALL_SEQUENCES columns

Name	Type	Description
relid	oid	OID of this sequence
schemaname	name	Name of the schema this sequence is in
relname	name	Name of this sequence
blks_read	bigint	Number of disk blocks read from the sequence

Name	Type	Description
blks_hit	bigint	Number of buffer hits in this sequence

14.3.144 PG_STATIO_ALL_TABLES

PG_STATIO_ALL_TABLES displays I/O statistics about all tables (including TOAST tables) in the current database.

Table 14-201 PG_STATIO_ALL_TABLES columns

Name	Type	Description
relid	oid	Table OID
schemaname	name	Schema name of the table
relname	name	Name of a table
heap_blks_read	bigint	Number of disks read from this table
heap_blks_hit	bigint	Number of buffer hits in this table
idx_blks_read	bigint	Number of disk blocks read from the index in this table
idx_blks_hit	bigint	Number of buffer hits in all indexes on this table
toast_blks_read	bigint	Number of disk blocks read from the TOAST table (if any) in this table
toast_blks_hit	bigint	Number of buffer hits in the TOAST table (if any) in this table
tidx_blks_read	bigint	Number of disk blocks read from the TOAST table index (if any) in this table
tidx_blks_hit	bigint	Number of buffer hits in the TOAST table index (if any) in this table

14.3.145 PG_STATIO_SYS_INDEXES

PG_STATIO_SYS_INDEXES displays the I/O status information about all system catalog indexes in the namespace.

Table 14-202 PG_STATIO_SYS_INDEXES columns

Name	Type	Description
relid	oid	Table OID for the index
indexrelid	oid	OID of this index
schemaname	name	Schema name for the index
relname	name	Name of the table for this index
indexrelname	name	Name of this index
idx_blks_read	bigint	Number of disk blocks read from the index
idx_blks_hit	bigint	Number of buffer hits in this index

14.3.146 PG_STATIO_SYS_SEQUENCES

PG_STATIO_SYS_SEQUENCES displays the I/O status information about all the system sequences in the namespace.

Table 14-203 PG_STATIO_SYS_SEQUENCES columns

Name	Type	Description
relid	oid	OID of this sequence
schemaname	name	Name of the schema this sequence is in
relname	name	Name of this sequence
blks_read	bigint	Number of disk blocks read from the sequence
blks_hit	bigint	Number of buffer hits in this sequence

14.3.147 PG_STATIO_SYS_TABLES

PG_STATIO_SYS_TABLES displays the I/O status information about all the system catalogs in the namespace.

Table 14-204 PG_STATIO_SYS_TABLES columns

Name	Type	Description
relid	oid	Table OID
schemaname	name	Schema name of the table
relname	name	Name of a table

Name	Type	Description
heap_blks_read	bigint	Number of disk blocks read from this table
heap_blks_hit	bigint	Number of buffer hits in this table
idx_blks_read	bigint	Number of disk blocks read from the index in this table
idx_blks_hit	bigint	Number of buffer hits in all indexes on this table
toast_blks_read	bigint	Number of disk blocks read from the TOAST table (if any) in this table
toast_blks_hit	bigint	Number of buffer hits in the TOAST table (if any) in this table
tidx_blks_read	bigint	Number of disk blocks read from the TOAST table index (if any) in this table
tidx_blks_hit	bigint	Number of buffer hits in the TOAST table index (if any) in this table

14.3.148 PG_STATIO_USER_INDEXES

PG_STATIO_USER_INDEXES displays the I/O status information about all the user relationship table indexes in the namespace.

Table 14-205 PG_STATIO_USER_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 this index is in
relname	name	Name of the table for this index
indexrelname	name	Name of this index
idx_blks_read	bigint	Number of disk blocks read from the index
idx_blks_hit	bigint	Number of buffer hits in this index

14.3.149 PG_STATIO_USER_SEQUENCES

PG_STATIO_USER_SEQUENCES displays the I/O status information about all the user relation table sequences in the namespace.

Table 14-206 PG_STATIO_USER_SEQUENCES columns

Name	Type	Description
relid	oid	OID of this sequence
schemaname	name	Name of the schema this sequence is in
relname	name	Name of this sequence
blks_read	bigint	Number of disk blocks read from the sequence
blks_hit	bigint	Cache hits in the sequence

14.3.150 PG_STATIO_USER_TABLES

PG_STATIO_USER_TABLES displays the I/O status information about all the user relation tables in the namespace.

Table 14-207 PG_STATIO_USER_TABLES columns

Name	Type	Description
relid	oid	Table OID
schemaname	name	Schema name of the table
relname	name	Name of a table
heap_blks_read	bigint	Number of disks read from this table
heap_blks_hit	bigint	Number of buffer hits in this table
idx_blks_read	bigint	Number of disk blocks read from the index in this table
idx_blks_hit	bigint	Number of buffer hits in all indexes on this table
toast_blks_read	bigint	Number of disk blocks read from the TOAST table (if any) in this table
toast_blks_hit	bigint	Number of buffer hits in the TOAST table (if any) in this table
tidx_blks_read	bigint	Number of disk blocks read from the TOAST table index (if any) in this table
tidx_blks_hit	bigint	Number of buffer hits in the TOAST table index (if any) in this table

14.3.151 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 14-208 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. It is equivalent to **debug_query_id**.
tid	bigint	Thread ID of the current thread
lwtid	integer	Lightweight thread ID of the current thread
ptid	integer	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 14-209**.
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.

The waiting statuses in the **wait_status** column are as follows:

Table 14-209 Waiting status list

Value	Description
none	Waiting for no event
acquire lock	Waiting for locking until the locking succeeds or times out
acquire lwlock	Waiting for a lightweight lock
wait io	Waiting for I/O completion
wait cmd	Waiting for network communication packet read to complete
wait pooler get conn	Waiting for pooler to obtain the connection

Value	Description
wait pooler abort conn	Waiting for pooler to terminate the connection
wait pooler clean conn	Waiting for 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 the data from a connected node. The thread is waiting for the 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*
wait wal sync	Waiting for the completion of wal log 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 storage or the CU in the column storage into the synchronization queue

Value	Description
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 steam plan ends
nodegroup destroy	Waiting for destroying the stream node group when the steam plan ends
wait active statement	Waiting for job execution under resource and load control.
wait global queue	Waiting for job execution. The job is queuing in the global queue.
wait respool queue	Waiting for job execution. The job is queuing in the resource pool.
wait ccn queue	Waiting for job execution. The job is queuing on the central coordinator node (CCN).
gtm connect	Waiting for connecting to GTM.
gtm get gxid	Wait for obtaining xids from GTM.
gtm get snapshot	Wait for obtaining transaction snapshots from GTM.
gtm begin trans	Waiting for GTM to start a transaction.
gtm commit trans	Waiting for GTM to commit a transaction.
gtm rollback trans	Waiting for GTM to roll back a transaction.
gtm create sequence	Waiting for GTM to create a sequence.
gtm alter sequence	Waiting for GTM to modify a sequence.
gtm get sequence val	Waiting for obtaining the next value of a sequence from GTM.

Value	Description
gtm set sequence val	Waiting for GTM to set a sequence value.
gtm drop sequence	Waiting for GTM to delete a sequence.
gtm rename sequece	Waiting for GTM to rename a sequence.
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.
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 - write file	The **Sort** operator is being executed. **write file** indicates that the **Sort** operator is writing data to disks.

Value	Description
Material | Material - write file	The **Material** operator is being executed. **write file** indicates that the **Material** operator is writing data to disks.
wait sync consumer next step	The consumer (receive end) synchronously waits for the next iteration.
wait sync producer next step	The producer (transmit end) synchronously waits for the next iteration.
wait agent release	The current agent is being released (supported by 8.1.2 and later versions).
wait stream task	The stream thread is waiting for being reused (supported by 8.1.2 and later versions).

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 14-210 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 XID
ProcArrayLock	Used to prevent concurrent access to or concurrent modification on the ProcArray shared array
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

wait_event	Description
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
MultiXactGenLock	Used to allocate a unique MultiXact ID in serial mode
MultiXactOffsetControl-Lock	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
SyncScanLock	Used to determine the start position of a relfilenode during heap scanning
NodeTableLock	Used to protect a shared structure that stores CN and DN 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
AsyncCtlLock	Used to prevent concurrent access to or concurrent modification on the sharing notification status

wait_event	Description
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
SerializableFinishedList-Lock	Used to prevent concurrent read/write or concurrent write/write on a shared linked list for completed serial transactions
SerializablePredicateLock-ListLock	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
CStoreCUCacheSweep-Lock	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
DfsConnectorCacheLock	Used to protect a global hash table where HDFS connection handles are cached
dummyServerInfoCache-Lock	Used to protect a global hash table where the information about computing Node Group connections is cached
ExtensionConnectorLi-bLock	Used to add 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
DfsUserLoginLock	Used to protect a global linked table where HDFS user information is stored
DfsSpaceCacheLock	Used to ensure that the IDs of files to be imported to an HDFS table increase monotonically

wait_event	Description
LsnXlogChkFileLock	Used to serially update the Xlog flush points for primary and standby servers recorded in a specific structure
GTMHostInfoLock	Used to prevent concurrent access to or concurrent modification on GTM host information
ReplicationSlotAllocation-Lock	Used to add a lock when a primary server allocates stream replication slots during primary-standby replication
ReplicationSlotControl-Lock	Used to prevent concurrent update of 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 CN side
WorkloadIoStatHashLock	Used to prevent concurrent access to or concurrent modification on a hash table that contains the I/O information of the current DN
WorkloadCGroupHash-Lock	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
WorkloadRecordLock	Used to prevent concurrent access to or concurrent modification on a hash table that contains requests received by CNs 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 where MPP is compatible with Oracle
OBSRuntimeLock	Used to obtain environment variables, for example, *GAUSSHOME*.

wait_event	Description
LLVMDumpIRLock	Used to export the assembly language for dynamically generating functions
LLVMParseIRLock	Used to compile and parse a finished IR function from the IR file at the start position of a query
RPNumberLock	Used by a DN on a computing Node Group to count the number of threads for a task where plans are being executed
ClusterRPLock	Used to control concurrent access on cluster load data maintained in a CCN of the cluster
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 table mapped to shared buffer
LockMgrLock	It is used to protect a common lock structure.
PredicateLockMgrLock	Used to protect 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

The following table describes the corresponding wait events when **wait_status** is **wait io**.

Table 14-211 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, mainly during database initialization.
ControlFileSyncUpdate	Flushes the **pg_control** file to a disk, mainly during database startup, checkpoint execution, and primary/standby verification.
ControlFileWrite	Writes to the **pg_control** file, mainly 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.
LockFileAddToDataDir-Read	Reads the **postmaster.pid** file.
LockFileAddToDataDir-Sync	Flushes the **postmaster.pid** file to a disk.
LockFileAddToDataDir-Write	Writes the PID information into the **postmaster.pid** file.
LockFileCreateRead	Read the LockFile file **%s.lock**.
LockFileCreateSync	Flushes the LockFile file **%s.lock** to a disk.

wait_event	Description
LockFileCreateWRITE	Writes the 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.
ReplicationSlotRestore-Sync	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 submission and restoration.
TwophaseFileSync	Flushes the **pg_twophase** file to a disk, mainly during two-phase transaction submission and restoration.
TwophaseFileWrite	Writes the **pg_twophase** file, mainly during two-phase transaction submission and restoration.

wait_event	Description
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.

The following table describes the corresponding wait events when **wait_status** is **acquire lock**.

Table 14-212 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.

wait_event	Description
cstore_freespace	Adds a lock to idle column-store space.
userlock	Adds a lock to a user.
advisory	Adds an advisory lock.

14.3.152 PG_TABLES

PG_TABLES displays access to each table in the database.

Table 14-213 PG_TABLES columns

Name	Type	Reference	Description
schemaname	name	**PG_NAMESPACE**.nspname	Name of the schema that contains the table
tablename	name	**PG_CLASS**.relname	Name of the table
tableowner	name	pg_get_userbyid(**PG_CLASS**.relowner)	Owner of the table
tablespace	name	**PG_TABLESPACE**.spcname	Tablespace that contains the table. The default value is null
hasindexes	boolean	**PG_CLASS**.relhasindex	Whether the table has (or recently had) an index. If it does, its value is **true**. Otherwise, its value is **false**.
hasrules	boolean	**PG_CLASS**.relhasrules	Whether the table has rules. If it does, its value is **true**. Otherwise, its value is **false**.
hastriggers	boolean	**PG_CLASS**.RELHASTRIGGERS	Whether the table has triggers. If it does, its value is **true**. Otherwise, its value is **false**.
tablecreator	name	pg_get_userbyid(**PG_OBJECT**.creator)	Table creator. If the creator has been deleted, no value is returned.
created	timestamp with time zone	**PG_OBJECT**.ctime	Time when the table was created.

Name	Type	Reference	Description
last_ddl_time	timestamp with time zone	**PG_OBJECT**.mtime	Last modification time of the table (that is, the last time that a DDL statement is executed on the table).

Example

Query all tables in a specified schema.

```
SELECT tablename FROM PG_TABLES WHERE schemaname = 'myschema';
   tablename
----------------
 inventory
 product
 sales_info
 test1
 mytable
 product_info
 customer_info
 newproducts
 customer_t1
(9 rows)
```

14.3.153 PG_TDE_INFO

PG_TDE_INFO displays the encryption information about the current cluster.

Table 14-214 PG_TDE_INFO columns

Name	Type	Description
is_encrypt	text	Whether the cluster is an encryption cluster ● **f**: Non-encryption cluster ● **t**: Encryption cluster
g_tde_algo	text	Encryption algorithm ● SM4-CTR-128 ● AES-CTR-128
remain	text	Reserved columns

Examples

Check whether the current cluster is encrypted, and check the encryption algorithm (if any) used by the current cluster.

```
SELECT * FROM PG_TDE_INFO;
 is_encrypt | g_tde_algo  | remain
------------+-------------+--------
 f          | AES-CTR-128 | remain
(1 row)
```

14.3.154 PG_TIMEZONE_ABBREVS

PG_TIMEZONE_ABBREVS displays all time zone abbreviations that can be recognized by the input routines.

Table 14-215 PG_TIMEZONE_ABBREVS columns

Name	Type	Description
abbrev	text	Time zone abbreviation
utc_offset	interval	Offset from UTC
is_dst	boolean	Whether the abbreviation indicates a daylight saving time (DST) zone. If it does, its value is **true**. Otherwise, its value is **false**.

14.3.155 PG_TIMEZONE_NAMES

PG_TIMEZONE_NAMES displays all time zone names that can be recognized by **SET TIMEZONE**, along with their associated abbreviations, UTC offsets, and daylight saving time statuses.

Table 14-216 PG_TIMEZONE_NAMES columns

Name	Type	Description
name	text	Name of the time zone
abbrev	text	Time zone name abbreviation
utc_offset	interval	Offset from UTC
is_dst	boolean	Whether DST is used. If it is, its value is **true**. Otherwise, its value is **false**.

14.3.156 PG_TOTAL_MEMORY_DETAIL

PG_TOTAL_MEMORY_DETAIL displays the memory usage of a certain node in the database.

Table 14-217 PG_TOTAL_MEMORY_DETAIL columns

Name	Type	Description
nodename	text	Node name

Name	Type	Description
memorytype	text	It can be set to any of the following values: ● **max_process_memory**: memory used by a GaussDB(DWS) cluster instance ● **process_used_memory**: memory used by a GaussDB(DWS) 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 column store ● **cstore_used_memory**: memory used for column store ● **max_sctpcomm_memory**: maximum memory allowed for the communication library ● **sctpcomm_used_memory**: memory used for the communication library ● **sctpcomm_peak_memory**: memory peak of the communication library ● **max_topsql_memory**: maximum memory that can be used by Top SQL to record historical job monitoring information ● **topsql_used_memory**: memory used by Top SQL to record historical job monitoring information ● **topsql_peak_memory**: memory peak of Top SQL to record historical job monitoring information ● **other_used_memory**: other used memory ● **gpu_max_dynamic_memory**: maximum GPU memory

Name	Type	Description
		• **gpu_dynamic_used_memory**: sum of the available GPU memory and temporary GPU memory • **gpu_dynamic_peak_memory**: maximum memory used for GPU • **pooler_conn_memory**: memory used for pooler connections • **pooler_freeconn_memory**: memory used for idle pooler connections • **storage_compress_memory**: memory used for column-store compression and decompression • **udf_reserved_memory**: memory reserved for the **UDF Worker** process • **mmap_used_memory**: memory used for **mmap**
memorymbytes	integer	Size of the used memory (MB)

14.3.157 PG_TOTAL_SCHEMA_INFO

PG_TOTAL_SCHEMA_INFO displays the storage usage of all schemas in each database. This view is valid only if use_workload_manager is set to **on**.

Column	Type	Description
schemaid	oid	Schema OID
schemaname	text	Schema name
databaseid	oid	Database OID
databasename	name	Database name
usedspace	bigint	Size of the permanent table storage space used by the schema, in bytes.
permspace	bigint	Upper limit of the permanent table storage space of the schema, in bytes.

14.3.158 PG_TOTAL_USER_RESOURCE_INFO

PG_TOTAL_USER_RESOURCE_INFO displays the resource usage of all users. Only administrators can query this view. This view is valid only if **use_workload_manager** is set to **on**.

Table 14-218 PG_TOTAL_USER_RESOURCE_INFO columns

Name	Type	Description
username	name	Username
used_memory	integer	Memory size used by a user, in (MB). ● DN: The memory used by users on the current DN is displayed. ● CN: The total memory usage of users on all DNs is displayed.
total_memory	integer	Memory used by the resource pool (MB). **0** indicates that the available memory is not limited and depends on the maximum memory available in the database (**max_dynamic_memory**). A calculation formula is as follows: total_memory = max_dynamic_memory * parent_percent * user_percent CN: The sum of maximum available memory on all DNs is displayed.
used_cpu	double precision	Number of CPU cores in use. Only the CPU usage of complex jobs in the non-default resource pool is collected, and the value is the CPU usage of the related cgroup.
total_cpu	integer	Total number of CPU cores of the Cgroup associated with a user on the node
used_space	bigint	Used permanent table storage space (unit: KB)
total_space	bigint	Available permanent table storage space (unit: KB) The value **-1** indicates no limit.
used_temp_space	bigint	Used temporary table storage space (unit: KB)
total_temp_space	bigint	Available temporary table storage space (unit: KB) The value **-1** indicates no limit.
used_spill_space	bigint	Space used for operator spill, in KB.
total_spill_space	bigint	Available space for operator spill, in KB. The value **-1** indicates no limit.

Name	Type	Description
read_kbytes	bigint	On a CN, it indicates total number of bytes read by a user's complex jobs on all DNs in the last 5 seconds. The unit is KB. On a DN, it indicates the total number of bytes read by a user's complex jobs from the instance startup time to the current time. The unit is KB.
write_kbytes	bigint	On a CN, it indicates total number of bytes written by a user's complex jobs on all DNs in the last 5 seconds. On a DN, it indicates the total number of bytes written by a user's complex jobs from the instance startup time to the current time. The unit is KB.
read_counts	bigint	CN: total number of read times of a user's complex jobs on all DNs in the last 5 seconds. DN: total number of read times of a user's complex jobs from the instance startup time to the current time.
write_counts	bigint	CN: total number of write times of a user's complex jobs on all DNs in the last 5 seconds. DN: total number of write times of a user's complex jobs from the instance startup time to the current time.
read_speed	double precision	On a CN, it indicates the average read rate of a user's complex jobs on a single DN in the last 5 seconds, in KB/s. On a DN, it indicates the average read rate of a user's complex jobs on the DN in the last 5 seconds, in KB/s.
write_speed	double precision	On a CN, it indicates the average write rate of a user's complex jobs on a single DN in the last 5 seconds, in KB/s. On a DN, it indicates the average write rate of a user's complex jobs on the DN in the last 5 seconds, in KB/s.

14.3.159 PG_USER

PG_USER displays information about users who can access the database.

Table 14-219 PG_USER columns

Name	Type	Description
usename	name	User name
usesysid	oid	ID of this user
usecreatedb	boolean	Whether the user has the permission to create databases
usesuper	boolean	whether the user is the initial system administrator with the highest rights.
usecatupd	boolean	whether the user can directly update system tables. Only the initial system administrator whose usesysid is 10 has this permission. It is not available for other users.
userepl	boolean	Whether the user has the permission to duplicate data streams
passwd	text	Encrypted user password. The value is displayed as ********.
valbegin	timestamp with time zone	Account validity start time; null if no start time
valuntil	timestamp with time zone	Password expiry time; null if no expiration
respool	name	Resource pool where the user is in
parent	oid	Parent user OID
spacelimit	text	The storage space of the permanent table.
tempspacelimit	text	The storage space of the temporary table.
spillspacelimit	text	The operator disk flushing space.
useconfig	text[]	Session defaults for run-time configuration variables
nodegroup	name	Name of the logical cluster associated with the user. If no logical cluster is associated, this column is left blank.

Example

Query the current database user list.

```
SELECT usename  FROM pg_user;
  usename
```

```
-----------
 dbadmin
 u1
 u2
 u3
(4 rows)
```

14.3.160 PG_USER_MAPPINGS

PG_USER_MAPPINGS displays 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.

Table 14-220 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 this 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, its value is keyword=value strings. Otherwise, its value is null.

14.3.161 PG_VIEWS

PG_VIEWS displays basic information about each view in the database.

Table 14-221 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

Name	Type	Reference	Description
definition	text	-	Definition of the view

Example

Query all the views in a specified schema.

```
SELECT * FROM pg_views WHERE schemaname = 'myschema';
 schemaname | viewname | viewowner |                                 definition
------------+----------+-----------+--------------------------------------------------------------------------------------
 myschema   | myview   | dbadmin   | SELECT  * FROM pg_tablespace WHERE (pg_tablespace.spcname = 
'pg_default'::name);
 myschema   | v1       | dbadmin   | SELECT  * FROM t1 WHERE (t1.c1 > 200);
(2 rows)
```

14.3.162 PG_WLM_STATISTICS

PG_WLM_STATISTICS displays information about workload management after the task is complete or the exception has been handled. This view has been discarded in 8.1.2.

Table 14-222 PG_WLM_STATISTICS columns

Name	Type	Description
statement	text	Statement executed for exception handling
block_time	bigint	Block time before the statement is executed
elapsed_time	bigint	Elapsed time when the statement is executed
total_cpu_time	bigint	Total time used by the CPU on the DN when the statement is executed for exception handling
qualification_time	bigint	Period when the statement checks the inclination ratio
cpu_skew_percent	integer	CPU usage skew on the DN 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 it is executed for exception handling • **pending**: The statement is waiting to be executed. • **running**: The statement is being executed. • **finished**: The execution is finished normally. • **abort**: The execution is unexpectedly terminated.

Name	Type	Description
action	text	Actions when statements are executed for exception handling ● **abort** indicates terminating the operation. ● **adjust** indicates executing the Cgroup adjustment operations. Currently, you can only perform the demotion operation. ● **finish** indicates that the operation is normally finished.
queryid	bigint	Internal query ID used for statement execution
threadid	bigint	ID of the backend thread

14.3.163 PGXC_BULKLOAD_PROGRESS

PGXC_BULKLOAD_PROGRESS displays the progress of the service import. Only GDS common files can be imported. This view is accessible only to users with system administrators rights.

Table 14-223 PGXC_BULKLOAD_PROGRESS columns

Name	Type	Description
session_id	bigint	GDS session ID
query_id	bigint	Query ID. It is equivalent to **debug_query_id**.
query	text	Query statement
progress	text	Progress percentage

14.3.164 PGXC_BULKLOAD_STATISTICS

PGXC_BULKLOAD_STATISTICS displays real-time statistics about service execution, such as GDS, COPY, and \COPY, on a CN. This view summarizes the real-time execution status of import and export services that are being executed on each node in the current cluster. In this way, you can monitor the real-time progress of import and export services and locate performance problems.

Columns in **PGXC_BULKLOAD_STATISTICS** are the same as those in **PG_BULKLOAD_STATISTICS**. This is because **PGXC_BULKLOAD_STATISTICS** is essentially the summary result of querying **PG_BULKLOAD_STATISTICS** on each node in the cluster.

This view is accessible only to users with system administrators rights.

Table 14-224 PGXC_BULKLOAD_STATISTICS columns

Name	Type	Description
node_name	text	Node name
db_name	text	Database name
query_id	bigint	Query ID. It is equivalent to **debug_query_id**.
tid	bigint	ID of the current thread
lwtid	integer	Lightweight thread ID
session_id	bigint	GDS session ID
direction	text	Service type. The options are **gds to file**, **gds from file**, **gds to pipe**, **gds from pipe**, **copy from**, and **copy to**.
query	text	Query statement
address	text	Location of the foreign table used for data import and export
query_start	timestamp with time zone	Start time of data import or export
total_bytes	bigint	Total size of data to be processed This parameter is specified only when a GDS common file is to be imported and the record in the row comes from a CN. Otherwise, left this parameter unspecified.
phase	text	Current phase. The options are **INITIALIZING**, **TRANSFER_DATA**, and **RELEASE_RESOURCE**.
done_lines	bigint	Number of lines that have been transferred
done_bytes	bigint	Number of bytes that have been transferred

14.3.165 PGXC_COLUMN_TABLE_IO_STAT

PGXC_COLUMN_TABLE_IO_STAT provides I/O statistics of all column-store tables of the database on all CNs and DNs in the cluster. Except the **nodename** column of the name type added in front of each row, the names, types, and sequences of other columns are the same as those in the **GS_COLUMN_TABLE_IO_STAT** view. For details about the columns, see **Table 14-225**.

Table 14-225 GS_COLUMN_TABLE_IO_STAT columns

Name	Type	Description
schemaname	name	Namespace of a table
relname	name	Table name
heap_read	bigint	Number of blocks logically read in the heap
heap_hit	bigint	Number of block hits in the heap
idx_read	bigint	Number of blocks logically read in the index
idx_hit	bigint	Number of block hits in the index
cu_read	bigint	Number of logical reads in the Compression Unit
cu_hit	bigint	Number of hits in the Compression Unit
cidx_read	bigint	Number of indexes logically read in the Compression Unit
cidx_hit	bigint	Number of index hits in the Compression Unit

14.3.166 PGXC_COMM_CLIENT_INFO

PGXC_COMM_CLIENT_INFO stores the client connection information of all nodes. (You can query this view on a DN to view the information about the connection between the CN and DN.)

Table 14-226 PGXC_COMM_CLIENT_INFO columns

Name	Type	Description
node_name	text	Current node name.
app	text	Client application name
tid	bigint	Thread ID of the current thread.
lwtid	integer	Lightweight thread ID of the current thread.
query_id	bigint	Query ID. It is equivalent to **debug_query_id**.
socket	integer	It is displayed if the connection is a physical connection.
remote_ip	text	Peer node IP address.
remote_port	text	Peer node port.

Name	Type	Description
logic_id	integer	If the connection is a logical connection, **sid** is displayed. If **-1** is displayed, the current connection is a physical connection.

14.3.167 PGXC_COMM_DELAY

PGXC_COMM_STATUS displays the communication library delay status for all the DNs.

Table 14-227 PGXC_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
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. Its unit is 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. Its unit is microsecond.
max_delay	integer	Maximum delay of the current physical connection within 1 minute. The unit is microsecond.

14.3.168 PGXC_COMM_RECV_STREAM

PG_COMM_RECV_STREAM displays the receiving stream status of the communication libraries for all the DNs.

Table 14-228 PGXC_COMM_RECV_STREAM columns

Name	Type	Description
node_name	text	Node name
local_tid	bigint	ID of the thread using this stream

Name	Type	Description
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	Current status of the stream ● **UNKNOWN**: The logical connection is unknown. ● **READY**: The logical connection is ready. ● **RUN**: The logical connection receives packets normally. ● **HOLD**: The logical connection is waiting to receive packets. ● **CLOSED**: The logical connection is closed. ● **TO_CLOSED**: The logical connection is to 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. The unit is byte.
time	bigint	Current life cycle service duration of the stream. The unit is ms.
speed	bigint	Average receiving rate of the stream. The unit is byte/s.
quota	bigint	Current communication quota value of the stream. The unit is Byte.
buff_usize	bigint	Current size of the data cache of the stream. The unit is byte.

14.3.169 PGXC_COMM_SEND_STREAM

PGXC_COMM_SEND_STREAM displays the sending stream status of the communication libraries for all the DNs.

Table 14-229 PGXC_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	Current status of the stream ● **UNKNOWN**: The logical connection 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 is to 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
send_bytes	bigint	Total data volume sent by the stream. The unit is Byte.
time	bigint	Current life cycle service duration of the stream. The unit is ms.
speed	bigint	Average sending rate of the stream. The unit is Byte/s.
quota	bigint	Current communication quota value of the stream. The unit is Byte.
wait_quota	bigint	Extra time generated when the stream waits the quota value. The unit is ms.

14.3.170 PGXC_COMM_STATUS

PGXC_COMM_STATUS displays the communication library status for all the DNs.

Table 14-230 PGXC_COMM_STATUS columns

Name	Type	Description
node_name	text	Node name
rxpck/s	integer	Receiving rate of the communication library on a node. The unit is byte/s.
txpck/s	integer	Sending rate of the communication library on a node. The unit is byte/s.
rxkB/s	bigint	Receiving rate of the communication library on a node. The unit is KB/s.
txkB/s	bigint	Sending rate of the communication library on a node. The unit is KB/s.
buffer	bigint	Size of the buffer of the Cmailbox.
memKB(libcomm)	bigint	Communication memory size of the **libcomm** process, in KB.
memKB(libpq)	bigint	Communication memory size of the **libpq** process, in KB.
%USED(PM)	integer	Real-time usage of the postmaster thread.
%USED (sflow)	integer	Real-time usage of the **gs_sender_flow_controller** thread.
%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.

14.3.171 PGXC_COMM_QUERY_SPEED

PGXC_COMM_QUERY_SPEED displays traffic information about all queries on all nodes.

Table 14-231 PGXC_COMM_QUERY_SPEED columns

Name	Type	Description
node_name	text	Node name

Name	Type	Description
query_id	bigint	**debug_query_id** corresponding to the stream
rxkB/s	bigint	Receiving rate of the query stream (unit: byte/s)
txkB/s	bigint	Sending rate of the query stream (unit: byte/s)
rxkB	bigint	Total received data of the query stream (unit: byte)
txkB	bigint	Total sent data of the query stream (unit: byte)
rxpck/s	bigint	Packet receiving rate of the query (unit: packets/s)
txpck/s	bigint	Packet sending rate of the query (Unit: packets/s)
rxpck	bigint	Total number of received packets of the query
txpck	bigint	Total number of sent packets of the query

14.3.172 PGXC_DEADLOCK

PGXC_DEADLOCK displays lock wait information generated due to distributed deadlocks.

Currently, **PGXC_DEADLOCK** collects only lock wait information about locks whose **locktype** is **relation**, **partition**, **page**, **tuple**, or **transactionid**.

Table 14-232 PGXC_DEADLOCK columns

Name	Type	Description
locktype	text	Type of the locked object
nodename	name	Name of the node where the locked object resides
dbname	name	Name of the database where the locked object resides The value is **NULL** if the locked object is a transaction.
nspname	name	Name of the namespace of the locked object
relname	name	Name of the relation targeted by the lock The value is **NULL** if the object is not a relation or part of a relation.

Name	Type	Description
partname	name	Name of the partition targeted by the lock The value is **NULL** if the locked object is not a partition.
page	integer	Number of the page targeted by the lock The value is **NULL** if the locked object is neither a page nor a tuple.
tuple	smallint	Number of the tuple targeted by the lock The value is **NULL** if the locked object is not a tuple.
transactionid	xid	ID of the transaction targeted by the lock The value is **NULL** if the locked object is not a transaction.
waitusername	name	Name of the user who waits for the lock
waitgxid	xid	ID of the transaction that waits for the lock
waitxactstart	timestamp with time zone	Start time of the transaction that waits for the lock
waitqueryid	bigint	Latest query ID of the thread that waits for the lock
waitquery	text	Latest query statement of the thread that waits for the lock
waitpid	bigint	ID of the thread that waits for the lock
waitmode	text	Mode of the waited lock
holdusername	name	Name of the user who holds the lock
holdgxid	xid	ID of the transaction that holds the lock
holdxactstart	timestamp with time zone	Start time of the transaction that holds the lock
holdqueryid	bigint	Latest query ID of the thread that holds the lock
holdquery	text	Latest query statement of the thread that holds the lock
holdpid	bigint	ID of the thread that holds the lock
holdmode	text	Mode of the held lock

14.3.173 PGXC_GET_STAT_ALL_TABLES

PGXC_GET_STAT_ALL_TABLES displays information about insertion, update, and deletion operations on tables and the dirty page rate of tables.

Before running **VACUUM FULL** on a system catalog with a high dirty page rate, ensure that no user is performing operations on it. You are advised to run **VACUUM FULL** to tables (excluding system catalogs) whose dirty page rate exceeds 80% or run it based on service scenarios.

Table 14-233 PGXC_GET_STAT_ALL_TABLES columns

Name	Type	Description
relid	oid	Table OID
relname	name	Table name
schemaname	name	Schema name of the table
n_tup_ins	numeric	Number of inserted tuples
n_tup_upd	numeric	Number of updated tuples
n_tup_del	numeric	Number of deleted tuples
n_live_tup	numeric	Number of live tuples
n_dead_tup	numeric	Number of dead tuples
dirty_page_rate	numeric(5,2)	Dirty page rate (%) of a table

GaussDB(DWS) also provides the **pgxc_get_stat_dirty_tables(int dirty_percent, int n_tuples)** and **pgxc_get_stat_dirty_tables(int dirty_percent, int n_tuples, text schema)** functions to quickly filter out tables whose dirty page rate is greater than **dirty_percent**, number of dead tuples is greater than **n_tuples**, and schema name is **schema**.

For details, see **Other Functions**.

Examples

Use the view **PGXC_GET_STAT_ALL_TABLES** to query the tables whose dirty page rate is greater than 30%.

```
SELECT * FROM PGXC_GET_STAT_ALL_TABLES WHERE dirty_page_rate>30;
 relid |         relname          | schemaname | n_tup_ins | n_tup_upd | n_tup_del | n_live_tup | n_dead_tup |
dirty_page_rate
-------+--------------------------+------------+-----------+-----------+-----------+------------+------------
+-----------------
  2840 | pg_toast_2619            | pg_toast   |      7415 |         0 |      7415 |          0 |        291 |           88.00
  9001 | pgxc_class               | pg_catalog |     56331 |         3 |     56285 |         54 |        143 |           72.59
 53860 | reason                   | dbadmin    |         9 |        19 |         0 |          9 |         19 |           67.86
  9025 | pg_object                | pg_catalog |    112858 |   1179707 |    112619 |        246 |        429 |
63.56
  9015 | pgxc_node                | pg_catalog |        15 |        24 |         0 |         15 |         24 |           61.54
  2606 | pg_constraint            | pg_catalog |        78 |         0 |        42 |         36 |         42 |           53.85
  1260 | pg_authid              | pg_catalog |         6 |         6 |         0 |          6 |          6 |           50.00
(7 rows)
```

You can also use the **pgxc_get_stat_dirty_tables** function to query tables whose dirty page rate is greater than 10% and number of dirty data rows is greater than 1000.

```
SELECT a.schemaname,a.relname,pg_size_pretty(pg_table_size(b.oid)),a.dirty_page_rate FROM
pgxc_get_stat_dirty_tables(10,1000) a,pg_catalog.pg_class b WHERE a.relname = b.relname order by
pg_table_size(b.oid) desc;
 schemaname |   relname    | pg_size_pretty | dirty_page_rate
------------+--------------+----------------+-----------------
 pg_catalog | pg_attribute | 2792 KB        |           12.09
 pg_catalog | pg_class     | 568 KB         |           15.36
 pg_catalog | pg_type      | 368 KB         |           12.17
(3 rows)
```

14.3.174 PGXC_GET_STAT_ALL_PARTITIONS

PGXC_GET_STAT_ALL_PARTITIONS displays information about insertion, update, and deletion operations on partitions of partitioned tables and the dirty page rate of tables.

The statistics of this view depend on the **ANALYZE** operation. To obtain the most accurate information, perform the **ANALYZE** operation on the partitioned table first.

Table 14-234 PGXC_GET_STAT_ALL_PARTITIONS columns

Name	Type	Description
relid	oid	Table OID
partid	oid	Partition OID
schemaname	name	Schema name of the table
relname	name	Table name
partname	name	Partition name
n_tup_ins	numeric	Number of inserted tuples
n_tup_upd	numeric	Number of updated tuples
n_tup_del	numeric	Number of deleted tuples
n_live_tup	numeric	Number of live tuples
n_dead_tup	numeric	Number of dead tuples
page_dirty_rate	numeric(5,2)	Dirty page rate (%) of a table

Examples

Run the following command to query partition tables whose dirty page rate is greater than 30%:

```
SELECT * FROM PGXC_GET_STAT_ALL_PARTITIONS WHERE dirty_page_rate>30;
 relid | partid |   schemaname    |      relname       | partname | n_tup_ins | n_tup_upd | n_tup_del | n_live_tup
| n_dead_tup | dirty_page_rate
-------+--------+-----------------+--------------------+----------+-----------+-----------+-----------+------------
+------------+-----------------
 58320 |  58626 | schema_subquery | store_hash_par     | p1       |         2 |         0 |         2 |          0 |          2
|          100.00
 58430 |  58706 | schema_subquery | store_hash_par_mor | p4       |         1 |         1 |         1 |          0 |
2 |          100.00
 58320 |  58644 | schema_subquery | store_hash_par     | p1       |         3 |         0 |         3 |          0 |          3
|          100.00
 58430 |  58770 | schema_subquery | store_hash_par_mor | p4       |         1 |         1 |         1 |          0 |
2 |          100.00
 58320 |  58643 | schema_subquery | store_hash_par     | p1       |         2 |         0 |         2 |          0 |          2
|          100.00
 58320 |  58625 | schema_subquery | store_hash_par     | p1       |         2 |         0 |         2 |          0 |          2
|          100.00
 58320 |  58579 | schema_subquery | store_hash_par     | p1       |         2 |         0 |         2 |          0 |          2
|          100.00
 58320 |  58619 | schema_subquery | store_hash_par     | p1       |         3 |         0 |         3 |          0 |          3
|          100.00
 58320 |  58627 | schema_subquery | store_hash_par     | p1       |         4 |         0 |         4 |          0 |          4
|          100.00
 58320 |  58657 | schema_subquery | store_hash_par     | p1       |         3 |         0 |         3 |          0 |          3
|          100.00
(10 rows)
```

14.3.175 PGXC_GET_TABLE_SKEWNESS

PGXC_GET_TABLE_SKEWNESS displays the data skew on tables in the current database. Only the system administrator or the preset role **gs_role_read_all_stats** can access this view.

Table 14-235 PGXC_GET_TABLE_SKEWNESS columns

Name	Type	Description
schemaname	name	Schema name of a table
tablename	name	Name of a table
totalsize	numeric	Total size of a table, in bytes
avgsize	numeric(1000, 0)	Average table size (total table size divided by the number of DNs), which is the ideal size of tables distributed on each DN
maxratio	numeric(10,3)	Ratio of the maximum table size on a single DN to **avgsize**.
minratio	numeric(10,3)	Ratio of the minimum table size on a single DN to **avgsize**.
skewsize	bigint	Table skew rate (the maximum table size on a single DN minus the minimum table size on a single DN)
skewratio	numeric(10,3)	Table skew rate (skewsize/avgsize)

Name	Type	Description
skewstddev	numeric(1000, 0)	Standard deviation of table distribution (For two tables of the same size, a larger deviation indicates a more severe skew.)

Examples

Run the following command to query the data skews of all tables in the database (the number of tables in the database is less than 10,000):

```
SELECT * FROM pgxc_get_table_skewness ORDER BY totalsize DESC;
 schemaname |        tablename        | totalsize | avgsize | maxratio | minratio | skewsize | skewratio |
skewstddev
------------+-------------------------+-----------+---------+----------+----------+----------+-----------+------------
 dbadmin    | reason                  |    147456 |   49152 |     .333 |     .333 |        0 |     0.000 |          0
 tpcds      | reason_t2               |     73728 |   24576 |     .556 |    0.000 |    40960 |      .556 |      21674
 dbadmin    | reason_bk               |     65536 |   21845 |     .500 |    0.000 |    32768 |      .500 |      18919
 tsearch    | pgweb                   |     49152 |   16384 |     .333 |     .333 |        0 |     0.000 |          0
 dbadmin    | student                 |     40960 |   13653 |     .400 |     .200 |     8192 |      .200 |       4730
 tsearch    | ts_zhparser             |     40960 |   13653 |     .400 |     .200 |     8192 |      .200 |       4730
 dbms_om    | gs_wlm_session_info     |     24576 |    8192 |     .333 |     .333 |        0 |     0.000 |          0
 dbms_om    | gs_wlm_ec_operator_info |     24576 |    8192 |     .333 |     .333 |        0 |     0.000 |          0
 dbms_om    | gs_wlm_operator_info    |     24576 |    8192 |     .333 |     .333 |        0 |     0.000 |          0
(9 rows)
```

If the number of tables in the database is more than 10,000, do not use the **PGXC_GET_TABLE_SKEWNESS** view because it takes a long time (hours) to query the entire database for skewed columns. You are advised to refer to the definition of the **PGXC_GET_TABLE_SKEWNESS** view and use the **table_distribution()** function to define the output. This optimizes the calculation by reducing the output columns. An example is shown as follows:

```
SELECT schemaname,tablename,max(dnsize) AS maxsize, min(dnsize) AS minsize
FROM pg_catalog.pg_class c
INNER JOIN pg_catalog.pg_namespace n ON n.oid = c.relnamespace
INNER JOIN pg_catalog.table_distribution() s ON s.schemaname = n.nspname AND s.tablename =
c.relname
INNER JOIN pg_catalog.pgxc_class x ON c.oid = x.pcrelid AND x.pclocatortype = 'H'
GROUP BY schemaname,tablename;
```

14.3.176 PGXC_GTM_SNAPSHOT_STATUS

PGXC_GTM_SNAPSHOT_STATUS displays transaction information on the current GTM.

Table 14-236 PGXC_GTM_SNAPSHOT_STATUS columns

Name	Type	Description
xmin	xid	Minimum ID of the running transactions
xmax	xid	ID of the transaction next to the executed transaction with the maximum ID
csn	integer	Sequence number of the transaction to be committed

Name	Type	Description
oldestxmin	xid	Minimum ID of the executed transactions
xcnt	integer	Number of the running transactions
running_xids	text	IDs of the running transactions

14.3.177 PGXC_INSTANCE_TIME

PGXC_INSTANCE_TIME displays the running time of processes on each node in the cluster and the time consumed in each execution phase. Except the **node_name** column, the other columns are the same as those in the **PV_INSTANCE_TIME** view. Only the system administrator or the preset role **gs_role_read_all_stats** can access this view.

Table 14-237 PGXC_INSTANCE_TIME columns

Name	Type	Description
node_name	text	Node name
stat_id	integer	Type ID
stat_name	text	Name of the runtime type
value	bigint	Running time value

14.3.178 PGXC_LOCKWAIT_DETAIL

PGXC_LOCKWAIT_DETAIL displays detailed information about the lock wait hierarchy on each node in a cluster. If a node has multiple lock wait levels, the entire lock waiting hierarchy is displayed in sequence.

This view is supported only by clusters of version 8.1.3.200 or later.

Table 14-238 PGXC_LOCKWAIT_DETAIL columns

Name	Type	Description
level	integer	Level in the lock wait hierarchy. The value starts with 1 and increases by 1 when there is a wait relationship.
node_name	name	Node name, corresponding to the **node_name** column in the **pgxc_node** table.
lock_wait_hierarchy	text	Lock wait hierarchy , in the format of *Node name: Process ID->Waiting process ID->Waiting process ID->...*

Name	Type	Description
lock_type	text	Type of the locked object
database	oid	OID of the database where the locked target is
relation	oid	OID of the locked object relationship
page	integer	Page index in a relationship
tuple	smallint	Row number of a page.
virtual_xid	text	Virtual ID of a transaction
transaction_id	xid	Transaction ID
class_id	oid	OID of the system catalog that contains the object
obj_id	oid	OID of the object in its system catalog
obj_subid	smallint	Column number of a table
virtual_transaction	text	Virtual ID of the transaction holding or awaiting this lock
pid	bigint	ID of the thread holding or awaiting this lock
mode	text	Lock level
granted	boolean	Indicates whether a lock is held.
fastpath	boolean	Indicates whether to obtain a lock using FASTPATH.
wait_for_pid	bigint	ID of the thread where a lock conflict occurs.
conflict_mode	text	Level of the conflicted lock held by the thread where it is
query_id	bigint	ID of a query statement.
query	text	Query statement
application_name	text	Name of the application connected to the backend
backend_start	timestamp with time zone	Startup time of the backend process, that is, the time when the client connects to the server
xact_start	timestamp with time zone	Start time of the current transaction
query_start	timestamp with time zone	Start time of the active query

Name	Type	Description
state	text	Overall state of the backend

Examples

Step 1 Connect to the DN, start a transaction, and run the following command:

```
begin;select * from t1;
```

Step 2 Connect to the CN in another window and truncate table **t1**.

```
truncate t1;
```

In this case, truncation is blocked.

Step 3 Open another window to connect to the CN and run the **select * from pgxc_lockwait_detail;** command.

```
SELECT * FROM PGXC_LOCKWAIT_DETAIL;
level | node_name |              lock_wait_hierarchy              | lock_type | database |  relation  | page | tuple |
virtual_xid | transaction_id | class_id | obj_id | obj_subid | virtual_transaction |       p
id       |        mode         | granted | fastpath |  wait_for_pid   |  conflict_mode  |     query_id
|                    query                     | application_name |        backend_start         |
xact_start           |          query_start          |        state
-------+-----------+---------------------------------------------+-----------+----------+------------+------+-------
+-------------+----------------+----------+--------+-----------+---------------------+--------
---------+---------------------+---------+----------+-----------------+-----------------+-------------------
+----------------------------------------------+------------------+-------------------------------+--
----------------------------+------------------------------+---------------------
1 | datanode1 | datanode1:140378619314976                  | relation  |    16049 | 2147484411 |      |
|             |         673638 |          |        |           | 19/297              | 1403786
19314976 | AccessExclusiveLock | f       | f        | 140378619263840 | AccessShareLock | 73183493945504391
| TRUNCATE t1                     | coordinator1     | 2023-03-13 12:13:52.530602+08 | 2
023-03-13 14:52:16.1456+08   | 2023-03-13 14:52:16.148693+08 | active
2 | datanode1 | datanode1:140378619314976 -> 140378619263840 | relation  |    16049 | 2147484411 |
|       |             |                |          |        |           | 23/16067            | 1403786
19263840 | AccessShareLock     | t       | f        |                 |                 |                 0 | begin;select * from t1;
| gsql             | 2023-03-13 14:19:26.325602+08 | 2
023-03-13 14:52:12.042741+08 | 2023-03-13 14:52:12.042741+08 | idle in transaction
(2 rows)
```

----End

14.3.179 PGXC_INSTR_UNIQUE_SQL

PGXC_INSTR_UNIQUE_SQL displays the complete Unique SQL statistics of all CN nodes in the cluster.

Only the system administrator can access this view. The columns in this view are the same as those in the **GS_INSTR_UNIQUE_SQL** view. For details about columns in the view, see **Table 14-239**.

Table 14-239 GS_INSTR_UNIQUE_SQL columns

Name	Type	Description
node_name	name	Name of the CN that receives SQL statements

Name	Type	Description
node_id	integer	Node ID, which is the same as the value of **node_id** in the **pgxc_node** table
user_name	name	Username
user_id	oid	User ID
unique_sql_id	bigint	Normalized Unique SQL ID
query	text	Normalized SQL text
n_calls	bigint	Number of successful execution times
min_elapse_time	bigint	Minimum running time of the SQL statement in the database (unit: μs)
max_elapse_time	bigint	Maximum running time of SQL statements in the database (unit: μs)
total_elapse_time	bigint	Total running time of SQL statements in the database (unit: μs)
n_returned_rows	bigint	Row activity - Number of rows in the result set returned by the **SELECT** statement
n_tuples_fetched	bigint	Row activity - Randomly scan rows (column-store tables/foreign tables are not counted.)
n_tuples_returned	bigint	Row activity - Sequential scan rows (Column-store tables/foreign tables are not counted.)
n_tuples_inserted	bigint	Row activity - Inserted rows
n_tuples_updated	bigint	Row activity - Updated rows
n_tuples_deleted	bigint	Row activity - Deleted rows

Name	Type	Description
n_blocks_fetched	bigint	Block access times of the buffer, that is, physical read/I/O
n_blocks_hit	bigint	Block hits of the buffer, that is, logical read/cache
n_soft_parse	bigint	Number of soft parsing times (cache plan)
n_hard_parse	bigint	Number of hard parsing times (generation plan)
db_time	bigint	Valid DB execution time, including the waiting time and network sending time. If multiple threads are involved in query execution, the value of **DB_TIME** is the sum of **DB_TIME** of multiple threads (unit: μs).
cpu_time	bigint	CPU execution time, excluding the sleep time (unit: μs)
execution_time	bigint	SQL execution time in the query executor, DDL statements, and statements (such as Copy statements) that are not executed by the executor are not counted (unit: μs).
parse_time	bigint	SQL parsing time (unit: μs)
plan_time	bigint	SQL generation plan time (unit: μs)
rewrite_time	bigint	SQL rewriting time (unit: μs)
pl_execution_time	bigint	Execution time of the plpgsql procedural language function (unit: μs)

Name	Type	Description
pl_compilation_time	bigint	Compilation time of the plpgsql procedural language function (unit: μs)
net_send_time	bigint	Network time, including the time spent by the CN in sending data to the client and the time spent by the DN in sending data to the CN (unit: μs)
data_io_time	bigint	File I/O time (unit: μs)
first_time	timestamp with time zone	Time of the first SQL statement execution
last_time	timestamp with time zone	Time of the last SQL statement execution

14.3.180 PGXC_LOCK_CONFLICTS

PGXC_LOCK_CONFLICTS displays information about conflicting locks in the cluster.

When a lock is waiting for another lock or another lock is waiting for this one, a lock conflict occurs.

Currently, **PGXC_LOCK_CONFLICTS** collects only information about locks whose **locktype** is **relation**, **partition**, **page**, **tuple**, or **transactionid**.

Table 14-240 PGXC_LOCK_CONFLICTS columns

Name	Type	Description
locktype	text	Type of the locked object
nodename	name	Name of the node where the locked object resides
dbname	name	Name of the database where the locked object resides. The value is **NULL** if the locked object is a transaction.
nspname	name	Name of the namespace of the locked object
relname	name	Name of the relation targeted by the lock. The value is **NULL** if the object is not a relation or part of a relation.

Name	Type	Description
partname	name	Name of the partition targeted by the lock. The value is **NULL** if the locked object is not a partition.
page	integer	Number of the page targeted by the lock. The value is **NULL** if the locked object is neither a page nor a tuple.
tuple	smallint	Number of the tuple targeted by the lock. The value is **NULL** if the locked object is not a tuple.
transactionid	xid	ID of the transaction targeted by the lock. The value is **NULL** if the locked object is not a transaction.
username	name	Name of the user who applies for the lock
gxid	xid	ID of the transaction that applies for the lock
xactstart	timestamp with time zone	Start time of the transaction that applies for the lock
queryid	bigint	Latest query ID of the thread that applies for the lock
query	text	Latest query statement of the thread that applies for the lock
pid	bigint	ID of the thread that applies for the lock
mode	text	Lock mode
granted	boolean	● **TRUE** if the lock has been held ● **FALSE** if the lock is still waiting for another lock

14.3.181 PGXC_NODE_ENV

PGXC_NODE_ENV displays the environmental variables information about all nodes in a cluster.

Table 14-241 PGXC_NODE_ENV columns

Name	Type	Description
node_name	text	Names of all nodes in the cluster
host	text	Host names of all the nodes in the cluster
process	integer	Process IDs of all the nodes in the cluster

Name	Type	Description
port	integer	Port numbers of all the nodes in the cluster
installpath	text	Installation directory of all the nodes in the cluster
datapath	text	Data directory of all the nodes in the cluster
log_directory	text	Log directory of all the nodes in the cluster

14.3.182 PGXC_NODE_STAT_RESET_TIME

PGXC_NODE_STAT_RESET_TIME displays the time when statistics of each node in the cluster are reset. All columns except **node_name** are the same as those in the **GS_NODE_STAT_RESET_TIME** view. This view is accessible only to users with system administrators rights.

Table 14-242 PGXC_NODE_STAT_RESET_TIME columns

Name	Type	Description
node_name	text	Node name
reset_time	timestamp	Time when statistics on each node are reset

14.3.183 PGXC_OS_RUN_INFO

PGXC_OS_RUN_INFO displays the OS running status of each node in the cluster. All columns except **node_name** are the same as those in the **PV_OS_RUN_INFO** view. Only the system administrator or the preset role **gs_role_read_all_stats** can access this view.

Table 14-243 PGXC_OS_RUN_INFO columns

Name	Type	Description
node_name	text	Node name
id	integer	ID
name	text	Name of the OS status
value	numeric	Value of the OS status
comments	text	Remarks of the OS status
cumulative	boolean	Whether the value of the OS status is cumulative

14.3.184 PGXC_OS_THREADS

PGXC_OS_THREADS displays thread status information under all normal nodes in the current cluster.

Table 14-244 PGXC_OS_THREADS columns

Name	Type	Description
node_name	text	All normal node names in the cluster
pid	bigint	IDs of running threads among all the normal node processes in the current cluster
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

14.3.185 PGXC_PREPARED_XACTS

PGXC_PREPARED_XACTS displays the two-phase transactions in the **prepared** phase.

Table 14-245 PGXC_PREPARED_XACTS columns

Name	Type	Description
pgxc_prepared_xact	text	Two-phase transactions in **prepared** phase

14.3.186 PGXC_REDO_STAT

PGXC_REDO_STAT displays statistics on redoing Xlogs of each node in the cluster. All columns except **node_name** are the same as those in the **PV_REDO_STAT** view. Only the system administrator or the preset role **gs_role_read_all_stats** can access this view.

Table 14-246 PGXC_REDO_STAT columns

Name	Type	Description
node_name	text	Node name
phywrts	bigint	Number of physical writes
phyblkwrt	bigint	Number of physical write blocks

Name	Type	Description
writetim	bigint	Time taken for physical writes
avgiotim	bigint	Average time taken per write
lstiotim	bigint	Time taken for the last write
miniotim	bigint	Minimum time taken for a write
maxiowtm	bigint	Maximum time taken for a write

14.3.187 PGXC_REL_IOSTAT

PGXC_REL_IOSTAT displays statistics on disk read and write of each node in the cluster. All columns except **node_name** are the same as those in the **GS_REL_IOSTAT** view. This view is accessible only to users with system administrators rights.

Table 14-247 PGXC_REL_IOSTAT columns

Name	Type	Description
node_name	text	Node name
phyrds	bigint	Number of disk reads
phywrts	bigint	Number of disk writes
phyblkrd	bigint	Number of read pages
phyblkwrt	bigint	Number of written pages

14.3.188 PGXC_REPLICATION_SLOTS

PGXC_REPLICATION_SLOTS displays the replication information of DNs in the cluster. All columns except **node_name** are the same as those in the **PG_REPLICATION_SLOTS** view. This view is accessible only to users with system administrators rights.

Table 14-248 PGXC_REPLICATION_SLOTS columns

Name	Type	Description
node_name	text	Node name
slot_name	text	Name of a replication node
plugin	name	Name of the output plug-in of the logical replication slot
slot_type	text	Type of a replication node

Name	Type	Description
datoid	oid	OID of the database on the replication node
database	name	Name of the database on the replication node
active	boolean	Whether the replication node is active
xmin	xid	Transaction ID of the replication node
catalog_xmin	text	ID of the earliest-decoded transaction corresponding to the logical replication slot
restart_lsn	text	Xlog file information on the replication node
dummy_standby	boolean	Whether the replication node is the dummy standby node

14.3.189 PGXC_RESPOOL_RUNTIME_INFO

PGXC_RESPOOL_RUNTIME_INFO displays the running information about all resource pool jobs on all CNs.

Table 14-249 PGXC_RESPOOL_RUNTIME_INFO columns

Name	Type	Description
nodename	name	CN name
nodegroup	name	Name of the logical cluster of the resource pool. The default value is **installation**
rpname	name	Resource pool name
ref_count	int	Number of jobs referenced by resource pools. The number is counted regardless of whether a job is controlled by a resource pool.
fast_run	int	Number of running jobs in the fast lane of the resource pool
fast_wait	int	Number of jobs queued in the fast lane of the resource pool
slow_run	int	Number of running jobs in the slow lane of the resource pool
slow_wait	int	Number of jobs queued in the slow lane of the resource pool

14.3.190 PGXC_RESPOOL_RESOURCE_INFO

PGXC_RESPOOL_RESOURCE_INFO displays the real-time monitoring information about the resource pools on all instances.

NOTE

On a DN, it only displays the monitoring information of the logical cluster that the DN belongs to.

Table 14-250 PGXC_RESPOOL_RESOURCE_INFO columns

Name	Type	Description
nodename	name	Instance name, including CNs and DNs.
nodegroup	name	Name of the logical cluster of the resource pool. The default value is **installation**.
rpname	name	Resource pool name
cgroup	name	Name of the Cgroup associated with the resource pool
ref_count	int	Number of jobs referenced by the resource pool. The number is counted regardless of whether the job is controlled by the resource pool. This parameter is valid only on CNs.
fast_run	int	Number of running jobs in the fast lane of the resource pool. This parameter is valid only on CNs.
fast_wait	int	Number of jobs queued in the fast lane of the resource pool. This parameter is valid only on CNs.
fast_limit	int	Limit on the number of concurrent fast lane jobs in the resource pool. This parameter is valid only on CNs.
slow_run	int	Number of running jobs in the slow lane of the resource pool. This parameter is valid only on CNs.
slow_wait	int	Number of jobs queued in the slow lane of the resource pool. This parameter is valid only on CNs.
slow_limit	int	Limit on the number of concurrent slow lane jobs in the resource pool. This parameter is valid only on CNs.

Name	Type	Description
used_cpu	double	Average number of used CPUs of the resource pool in a 5s monitoring period. The value is accurate to two decimal places. ● On a DN, it indicates the number of CPUs used by the resource pool on the current DN. ● On a CN, it indicates the total CPU usage of resource pools on all DNs.
cpu_limit	int	It indicates the upper limit of available CPUs for resource pools. If the CPU time limit is specified, this parameter indicates the available CPUs for GaussDB(DWS). If the CPU usage limit is specified, this parameter indicates the available CPUs for associated Cgroups. ● On a DN, it indicates the upper limit of available CPUs for the resource pool on the current DN. ● On a CN, it indicates the total upper limit of available CPUs for resource pools on all DNs.
used_mem	int	Memory size used by the resource pool (unit: MB) ● On a DN, it indicates the memory usage of the resource pool on the current DN. ● On a CN, it indicates the total memory usage of resource pools on all DNs.
estimate_mem	int	Estimated memory used by the jobs running in the resource pools on the current CN. This parameter is valid only on CNs.
mem_limit	int	Upper limit of available memory for the resource pool (unit: MB) ● On a DN, it indicates the upper limit of available memory for the resource pool on the current DN. ● On a CN, it indicates the total upper limit of available memory for resource pools on all DNs.
read_kbytes	bigint	Number of logical read bytes in the resource pool within a 5s monitoring period (unit: KB) ● On a DN, it indicates the number of logical read bytes in the resource pool on the current DN. ● On a CN, it indicates the total logical read bytes of resource pools on all DNs.

Name	Type	Description
write_kbytes	bigint	Number of logical write bytes in the resource pool within a 5s monitoring period (unit: KB) ● On a DN, it indicates the number of logical write bytes in the resource pool on the current DN. ● On a CN, it indicates the total logical write bytes of resource pools on all DNs.
read_counts	bigint	Number of logical reads in the resource pool within a 5s monitoring period ● On a DN, it indicates the number of logical reads in the resource pool on the current DN. ● On a CN, it indicates the total number of logical reads in resource pools on all DNs.
write_counts	bigint	Number of logical writes in the resource pool within a 5s monitoring period ● On a DN, it indicates the number of logical writes in the resource pool on the current DN. ● On a CN, it indicates the total number of logical writes in resource pools on all DNs.
read_speed	double	Average rate of logical reads of the resource pool in a 5s monitoring period. ● On a DN, it indicates the logical read rate of the resource pool on the current DN. ● On a CN, it indicates the overall logical read rate of resource pools on all DNs.
write_speed	double	Average rate of logical writes of the resource pool in a 5s monitoring period ● On a DN, it indicates the logical write rate of the resource pool on the current DN. ● On a CN, it indicates the overall logical write rate of resource pools on all DNs.

14.3.191 PGXC_RESPOOL_RESOURCE_HISTORY

PGXC_RESPOOL_RESOURCE_HISTORY is used to query historical monitoring information about resource pools on all instances.

Table 14-251 PGXC_RESPOOL_RESOURCE_HISTORY columns

Name	Type	Description
nodename	name	Instance name, including CNs and DNs.
timestamp	timestamp	Persistence duration of resource pool monitoring information
nodegroup	name	Name of the logical cluster of the resource pool. The default value is **installation**.
rpname	name	Resource pool name
cgroup	name	Name of the Cgroup associated with the resource pool
ref_count	int	Number of jobs referenced by the resource pool. The number is counted regardless of whether the job is controlled by the resource pool. This parameter is valid only on CNs.
fast_run	int	Number of running jobs in the fast lane of the resource pool. This parameter is valid only on CNs.
fast_wait	int	Number of jobs queued in the fast lane of the resource pool. This parameter is valid only on CNs.
fast_limit	int	Limit on the number of concurrent fast lane jobs in the resource pool. This parameter is valid only on CNs.
slow_run	int	Number of running jobs in the slow lane of the resource pool. This parameter is valid only on CNs.
slow_wait	int	Number of jobs queued in the slow lane of the resource pool. This parameter is valid only on CNs.
slow_limit	int	Limit on the number of concurrent slow lane jobs in the resource pool. This parameter is valid only on CNs.
used_cpu	double	Average number of used CPUs of the resource pool in a 5s monitoring period. The value is accurate to two decimal places. ● On a DN, it indicates the number of CPUs used by the resource pool on the current DN. ● On a CN, it indicates the total CPU usage of resource pools on all DNs.

Name	Type	Description
cpu_limit	int	It indicates the upper limit of available CPUs for resource pools. If the CPU time limit is specified, this parameter indicates the available CPUs for GaussDB(DWS). If the CPU usage limit is specified, this parameter indicates the available CPUs for associated Cgroups. ● On a DN, it indicates the upper limit of available CPUs for the resource pool on the current DN. ● On a CN, it indicates the total upper limit of available CPUs for resource pools on all DNs.
used_mem	int	Memory used by the resource pool (unit: MB). ● On a DN, it indicates the memory usage of the resource pool on the current DN. ● On a CN, it indicates the total memory usage of resource pools on all DNs.
estimate_mem	int	Estimated memory used by the jobs running in the resource pools on the current CN. This parameter is valid only on CNs.
mem_limit	int	Upper limit of available memory for the resource pool (unit: MB) ● On a DN, it indicates the upper limit of available memory for the resource pool on the current DN. ● On a CN, it indicates the total upper limit of available memory for resource pools on all DNs.
read_kbytes	bigint	Number of logical read bytes in the resource pool within a 5s monitoring period (unit: KB) ● On a DN, it indicates the number of logical read bytes in the resource pool on the current DN. ● On a CN, it indicates the total logical read bytes of resource pools on all DNs.
write_kbytes	bigint	Number of logical write bytes in the resource pool within a 5s monitoring period (unit: KB) ● On a DN, it indicates the number of logical write bytes in the resource pool on the current DN. ● On a CN, it indicates the total logical write bytes of resource pools on all DNs.

Name	Type	Description
read_counts	bigint	Number of logical reads in the resource pool within a 5s monitoring period ● On a DN, it indicates the number of logical reads in the resource pool on the current DN. ● On a CN, it indicates the total number of logical reads in resource pools on all DNs.
write_counts	bigint	Number of logical writes in the resource pool within a 5s monitoring period ● On a DN, it indicates the number of logical writes in the resource pool on the current DN. ● On a CN, it indicates the total number of logical writes in resource pools on all DNs.
read_speed	double	Average rate of logical reads of the resource pool in a 5s monitoring period. ● On a DN, it indicates the logical read rate of the resource pool on the current DN. ● On a CN, it indicates the overall logical read rate of resource pools on all DNs.
write_speed	double	Average rate of logical writes of the resource pool in a 5s monitoring period ● On a DN, it indicates the logical write rate of the resource pool on the current DN. ● On a CN, it indicates the overall logical write rate of resource pools on all DNs.

14.3.192 PGXC_ROW_TABLE_IO_STAT

PGXC_ROW_TABLE_IO_STAT provides I/O statistics of all row-store tables of the database on all CNs and DNs in the cluster. Except the **nodename** column of the name type added in front of each row, the names, types, and sequences of other columns are the same as those in the **GS_ROW_TABLE_IO_STAT** view. For details about the columns, see **Table 14-252**.

Table 14-252 GS_ROW_TABLE_IO_STAT columns

Name	Type	Description
schemaname	name	Namespace of a table
relname	name	Table name
heap_read	bigint	Number of blocks logically read in the heap

Name	Type	Description
heap_hit	bigint	Number of block hits in the heap
idx_read	bigint	Number of blocks logically read in the index
idx_hit	bigint	Number of block hits in the index
toast_read	bigint	Number of blocks logically read in the **TOAST** table
toast_hit	bigint	Number of block hits in the **TOAST** table
tidx_read	bigint	Number of indexes logically read in the **TOAST** table
tidx_hit	bigint	Number of index hits in the **TOAST** table

14.3.193 PGXC_RUNNING_XACTS

PGXC_RUNNING_XACTS displays information about running transactions on each node in the cluster. The content is the same as that displayed in **PG_RUNNING_XACTS**.

Table 14-253 PGXC_RUNNING_XACTS columns

Name	Type	Description
handle	integer	Handle corresponding to the transaction in GTM
gxid	xid	Transaction ID
state	tinyint	Transaction status (**3**: prepared or **0**: starting)
node	text	Node name
xmin	xid	Minimum transaction ID **xmin** on the node
vacuum	boolean	Whether the current transaction is lazy vacuum
timeline	bigint	Number of database restart
prepare_xid	xid	Transaction ID in **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 from a CN to a DN

14.3.194 PGXC_SETTINGS

PGXC_SETTINGS displays the database running status of each node in the cluster. All columns except **node_name** are the same as those in the **PG_SETTINGS** view. This view is accessible only to users with system administrators rights.

Table 14-254 PGXC_SETTINGS columns

Name	Type	Description
node_name	text	Node name.
name	text	Parameter name.
setting	text	Current value of the parameter.
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 of parameter values including **internal**, **postmaster**, **sighup**, **backend**, **superuser**, and **user**.
vartype	text	Parameter type. It can be **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 data, the value of this column is null.
max_val	text	Maximum value of the parameter. If the parameter type is not numeric data, the value of this column is null.
enumvals	text[]	Valid values of an enum-typed 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.

Name	Type	Description
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.

14.3.195 PGXC_SESSION_WLMSTAT

PGXC_SESSION_WLMSTAT displays load management information about ongoing jobs executed on each CN in the current cluster.

Table 14-255 PGXC_SESSION_WLMSTAT columns

Name	Type	Description
nodename	name	Node name
datid	oid	OID of the database the backend is connected to
datname	name	Name of the database the backend is connected to
threadid	bigint	Thread ID of the backend
processid	integer	PID of the backend thread
usesysid	oid	OID of the user who logged into the backend
appname	text	Name of the application that is 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	Statement attributes ● **Ordinary**: default attribute of a statement before it is parsed by the database ● **Simple**: simple statements ● **Complicated**: complicated statements ● **Internal**: internal statement of the database
block_time	bigint	Pending duration of the statements by now (unit: s)
elapsed_time	bigint	Actual execution duration of the statements by now (unit: s)
total_cpu_time	bigint	Total CPU usage duration of the statement on the DN in the last period (unit: s)

Name	Type	Description
cpu_skew_percent	integer	CPU usage inclination ratio of the statement on the DN in the last period
statement_mem	integer	Estimated memory required for statement execution.
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	Cgroup currently used by the statement
status	text	Status of a statement, including: ● **pending** ● **running**: The statement is being executed. ● **finished**: The execution is finished normally. (If **enqueue** is set to **StoredProc** or **Transaction**, this state indicates that only some of the 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**
enqueue	text	Current queuing status of the statements, including: ● **Global**: global queuing. ● **Respool**: resource pool queuing. ● **CentralQueue**: queuing on the CCN ● **Transaction**: being in a transaction block ● **StoredProc**: being in a stored procedure ● **None**: not in a 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	Text of this backend's most recent query If **state** is **active**, this column shows the executing query. In all other states, it shows the last query that was executed.

Name	Type	Description
isplana	bool	In logical cluster mode, indicates whether a statement occupies the resources of other logical clusters. The default value is **f**, indicating that resources of other logical clusters are not occupied.
node_group	text	Logical cluster of the user running the statement
lane	text	Fast or slow lane for statement queries. ● **fast**: fast lane ● **slow**: slow lane ● **none**: not controlled

14.3.196 PGXC_STAT_ACTIVITY

PGXC_STAT_ACTIVITY displays information about the query performed by the current user on all the CNs in the current cluster.

Table 14-256 PGXC_STAT_ACTIVITY columns

Name	Type	Description
coorname	text	Name of the CN in the current cluster
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
lwtid	integer	Lightweight thread ID of the backend thread
usesysid	oid	OID of the user logging in to the backend
usename	name	Name of the user logging 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 that the client is connected via a Unix socket on the server machine or that 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 only be non-null 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, or **-1** if a Unix socket is used
backend_start	timestamp with time zone	Startup time of the backend process, that is, the time when the client connects to the server
xact_start	timestamp with time zone	Time when the current transaction was started, or **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 **state** is not **active**
state_change	timestamp with time zone	Time for the last status change
waiting	boolean	The value is **t** if the backend is currently waiting for a lock or node. Otherwise, the value is **f**.

Name	Type	Description
enqueue	text	Queuing status of a statement. Its value can be: ● **waiting in global queue**: The statement is in the global concurrent queues. ● **waiting in respool queue**: The statement is queuing in the resource pool. The scenarios are as follows: 1. When dynamic load balancing is enabled, the number of simple jobs exceeds the upper limit (**max_dop**) of concurrent jobs on the fast lane. 2. When dynamic load balancing is disabled, the number of simple jobs exceeds the upper limit (**max_dop**) of concurrent jobs on the fast lane or the number of complex jobs exceeds the upper limit of concurrent jobs on the slow lane. ● **waiting in ccn queue**: The job is in the CCN queue, which may be global memory queuing, slow lane memory queuing, or concurrent queuing. ● Empty or **no waiting queue**: The statement is running.

Name	Type	Description
state	text	Overall state 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** Only system administrators can view the session status of their accounts. The state information of other accounts is empty.
resource_pool	name	Resource pool used by the user
stmt_type	text	Type of a user statement
query_id	bigint	ID of a query
query	text	Text of this backend's most recent query If the **state** is **active**, this column shows the executing 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**)

Example

Run the following command to view blocked query statements.

```
SELECT datname,usename,state,query FROM PGXC_STAT_ACTIVITY WHERE waiting = true;
```

Check the working status of the snapshot thread.

```
SELECT application_name,backend_start,state_change,state,query FROM PGXC_STAT_ACTIVITY WHERE 
application_name='WDRSnapshot';
```

View the running query statements.

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

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 PGXC_STAT_ACTIVITY WHERE DATNAME='postgres';
 count
-------
     1
(1 row)
```

14.3.197 PGXC_STAT_BAD_BLOCK

PGXC_STAT_BAD_BLOCK displays statistics about page or CU verification failures after all nodes in a cluster are started.

Table 14-257 PGXC_STAT_BAD_BLOCK columns

Name	Type	Description
nodename	text	Node name
databaseid	integer	Database OID
tablespaceid	integer	Tablespace OID
relfilenode	integer	File object ID
forknum	integer	File type
error_count	integer	Number of verification failures
first_time	timestamp with time zone	Time of the first occurrence
last_time	timestamp with time zone	Time of the latest occurrence

14.3.198 PGXC_STAT_BGWRITER

PGXC_STAT_BGWRITER displays statistics on the background writer of each node in the cluster. All columns except **node_name** are the same as those in the **PG_STAT_BGWRITER** view. This view is accessible only to users with system administrators rights.

Table 14-258 PGXC_STAT_BGWRITER columns

Name	Type	Description
node_name	text	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	Time spent on writing files to the disk during checkpoints, in milliseconds
checkpoint_sync_time	double precision	Time spent on synchronizing data to the disk during checkpoints, in milliseconds
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 the backend
buffers_backend_fsync	bigint	Number of times that the backend has to execute **fsync**
buffers_alloc	bigint	Number of buffers allocated
stats_reset	timestamp with time zone	Time at which these statistics were reset

14.3.199 PGXC_STAT_DATABASE

PGXC_STAT_DATABASE displays the database status and statistics of each node in the cluster. All columns except **node_name** are the same as those in the **PG_STAT_DATABASE** view. This view is accessible only to users with system administrators rights.

Table 14-259 PGXC_STAT_DATABASE columns

Name	Type	Description
node_name	text	Node name.
datid	oid	Database OID.

Name	Type	Description
datname	name	Database name.
numbackends	integer	Number of backends currently connected to this database on the current node. This is the only column in this view that reflects the current state value. All columns return the accumulated value since the last reset.
xact_commit	bigint	Number of transactions in this database that have been committed on the current node.
xact_rollback	bigint	Number of transactions in this database that have been rolled back on the current node.
blks_read	bigint	Number of disk blocks read in this database on the current node.
blks_hit	bigint	Number of disk blocks found in the buffer cache on the current node, that is, the number of blocks hit in the cache. (This only includes hits in the GaussDB(DWS) buffer cache, not in the file system cache.)
tup_returned	bigint	Number of rows returned by queries in this database on the current node.
tup_fetched	bigint	Number of rows fetched by queries in this database on the current node.
tup_inserted	bigint	Number of rows inserted in this database on the current node.
tup_updated	bigint	Number of rows updated in this database on the current node.
tup_deleted	bigint	Number of rows deleted from this database on the current node.
conflicts	bigint	Number of queries canceled due to database recovery conflicts on the current node (conflicts occurring only on the standby server). For details, see **PG_STAT_DATABASE_CONFLICTS**.
temp_files	bigint	Number of temporary files created by this database on the current node. 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.

Name	Type	Description
temp_bytes	bigint	Size of temporary files written to this database on the current node. 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 in this database on the current node.
blk_read_time	double precision	Time spent reading data file blocks by backends in this database on the current node, in milliseconds.
blk_write_time	double precision	Time spent writing into data file blocks by backends in this database on the current node, in milliseconds.
stats_reset	timestamp with time zone	Time when the database statistics are reset on the current node.

14.3.200 PGXC_STAT_REPLICATION

PGXC_STAT_REPLICATION displays the log synchronization status of each node in the cluster. All columns except **node_name** are the same as those in the **PG_STAT_REPLICATION** view. Only users with system administrator permissions can access this view.

Table 14-260 PGXC_STAT_REPLICATION columns

Name	Type	Description
node_name	text	Node name
pid	bigint	PID of the thread
usesysid	oid	User system ID
usename	name	Username
application_name	text	Application name
client_addr	inet	Client address
client_hostname	text	Client name
client_port	integer	Client port number
backend_start	timestamp with time zone	Program start time

Name	Type	Description
state	text	Log replication state (catch-up or consistent streaming)
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)
sync_state	text	Synchronization state (asynchronous duplication, synchronous duplication, or potential synchronization)

14.3.201 PGXC_STAT_TABLE_DIRTY

PGXC_STAT_TABLE_DIRTY displays statistics about all the tables on all the CNs and DNs in the current cluster, and the dirty page rate of tables on a single CN or DN. This view is supported only by clusters of version 8.1.3 or later.

NOTE

The statistics of this view depend on the **ANALYZE** operation. To obtain the most accurate information, perform the **ANALYZE** operation on the table first.

Table 14-261 PGXC_STAT_TABLE_DIRTY columns

Name	Type	Description
nodename	text	Node name
schema	name	Schema name of the table
tablename	name	Table name
partname	name	Partition name of the partitioned table
last_vacuum	timestampwith time zone	Time of the last manual **VACUUM**
last_autovacuum	timestampwith time zone	Time of the last **AUTOVACUUM**

Name	Type	Description
last_analyze	timestampwith time zone	Time of the last manual **ANALYZE**
last_autoanalyze	timestampwith time zone	Time of the last **AUTOANALYZE**
vacuum_count	bigint	Number of times **VACUUM** operations
autovacuum_count	bigint	Number of **AUTOVACUUM** operations
analyze_count	bigint	Number of **ANALYZE** operations
autoanalyze_count	bigint	Number of **AUTOANALYZE_COUNT** operations
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 updated by **HOT** (no separate index update is required)
n_tup_change	bigint	Number of changed rows after **ANALYZE**
n_live_tup	bigint	Estimated number of live rows
n_dead_tup	bigint	Estimated number of dead rows
dirty_rate	bigint	Dirty page rate of a single CN or DN
last_data_changed	timestampwith time zone	Time when a table was last modified

Suggestion

- Before running **VACUUM FULL** on a system catalog with a high dirty page rate, ensure that no user is performing operations on it.
- You are advised to run **VACUUM FULL** to tables (excluding system catalogs) whose dirty page rate exceeds 80% or run it based on service scenarios.

Scenarios

1. Query the overall dirty page rate of all the user tables in a database.

```
select
    t1.schema,
    t1.tablename,
    t1.total_ins,
    t1.total_upd,
    t1.total_del,
    t1. total_tup_hot_upd,
    t1.total_change,
    t1.total_live,
    t1.total_dead,
    t1.total_dirty_rate,
    t1.max_dirty,
    t2.max_node,
    t1.min_dirty,
    t2.min_node
from
    (select
        a.schema,
        a.tablename,
        sum(a.n_tup_ins) as total_ins,
        sum(a.n_tup_upd) as total_upd,
        sum(a.n_tup_del) as total_del,
        sum(a.n_tup_hot_upd) as total_tup_hot_upd,
        sum(a.n_tup_change) as total_change,
        sum(a.n_live_tup) as total_live,
        sum(a.n_dead_tup) as total_dead,
        Round((total_dead / (total_dead + total_live + 0.0001) * 100),2) AS total_dirty_rate,
        max(a.dirty_rate) as max_dirty,
        min(a.dirty_rate) as min_dirty
    from pg_catalog.pgxc_stat_table_dirty a where a.partname is null and a.schema not in
('pg_toast','cstore','gs_logical_cluster','sys','dbms_om','information_schema','pg_catalog','dbms_output','
dbms_random','utl_raw','utl_raw dbms_sql','dbms_lob') group by a.tablename, a.schema
    ) t1,
    (select distinct
    tablename, schema,
    first_value(nodename) over(partition by tablename, schema order by dirty_rate) as min_node,
    first_value(nodename) over(partition by tablename, schema order by dirty_rate desc) as max_node
    from (select * from pg_catalog.pgxc_stat_table_dirty)) t2
where t1.tablename = t2.tablename and t1.schema = t2.schema;
```

2. Query the overall dirty page rate of all the tables (user tables and system catalogs) in a database.

```
select
    t1.schema,
    t1.tablename,
    t1.total_ins,
    t1.total_upd,
    t1.total_del,
    t1. total_tup_hot_upd,
    t1.total_change,
    t1.total_live,
    t1.total_dead,
    t1.total_dirty_rate,
    t1.max_dirty,
    t2.max_node,
    t1.min_dirty,
    t2.min_node
from
    (select
        a.schema,
        a.tablename,
        sum(a.n_tup_ins) as total_ins,
        sum(a.n_tup_upd) as total_upd,
        sum(a.n_tup_del) as total_del,
        sum(a.n_tup_hot_upd) as total_tup_hot_upd,
        sum(a.n_tup_change) as total_change,
        sum(a.n_live_tup) as total_live,
        sum(a.n_dead_tup) as total_dead,
        Round((total_dead / (total_dead + total_live + 0.0001) * 100),2) AS total_dirty_rate,
        max(a.dirty_rate) as max_dirty,
        min(a.dirty_rate) as min_dirty
    from pg_catalog.pgxc_stat_table_dirty a where a.partname is null group by a.tablename, a.schema
    ) t1,
    (select distinct
    tablename, schema,
    first_value(nodename) over(partition by tablename, schema order by dirty_rate) as min_node,
    first_value(nodename) over(partition by tablename, schema order by dirty_rate desc) as max_node
    from (select * from pg_catalog.pgxc_stat_table_dirty)) t2
where t1.tablename = t2.tablename and t1.schema = t2.schema;
```

3. Query all system catalogs in a database.

```
select * from pgxc_stat_table_dirty where schema in
('pg_toast','cstore','gs_logical_cluster','sys','dbms_om','information_schema','pg_catalog','dbms_output','
dbms_random','utl_raw','utl_raw dbms_sql','dbms_lob');
```

14.3.202 PGXC_SQL_COUNT

PGXC_SQL_COUNT displays the node-level and user-level statistics for the SQL statements of **SELECT**, **INSERT**, **UPDATE**, **DELETE**, and **MERGE INTO** and DDL, DML, and DCL statements of each CN in a cluster in real time, identifies query types with heavy load, and measures the capability of a cluster or a node to perform a specific type of query. You can calculate QPS based on the quantities and response time of the preceding types of SQL statements at certain time points. For example, **USER1 SELECT** is counted as **X1** at T1 and as **X2** at T2. The **SELECT** QPS of the user can be calculated as follows: (X2 – X1)/(T2 – T1). In this way, the system can draw cluster-user-level QPS curve graphs and determine cluster throughput, monitoring changes in the service load of each user. If there are drastic changes, the system can locate the specific statement type (such as **SELECT**, **INSERT**, **UPDATE**, **DELETE**, and **MERGE INTO**). You can also observe QPS curves to determine the time points when problems occur and then locate the problems using other tools. The curves provide a basis for optimizing cluster performance and locating problems.

Columns in the **PGXC_SQL_COUNT** view are the same as those in the **GS_SQL_COUNT** view. For details, see **Table 14-122**.

NOTE

If a **MERGE INTO** statement can be pushed down and a DN receives it, the statement will be counted on the DN and the value of the **mergeinto_count** column will increment by 1. If the pushdown is not allowed, the DN will receive an **UPDATE** or **INSERT** statement. In this case, the **update_count** or **insert_count** column will increment by 1.

14.3.203 PGXC_TABLE_CHANGE_STAT

PGXC_TABLE_CHANGE_STAT displays the changes of all tables of the database on all CNs in the cluster. Except the **nodename** column of the name type added in front of each row, the names, types, and sequences of other columns are the same as those in the **GS_TABLE_CHANGE_STAT** view.

Table 14-262 PGXC_TABLE_CHANGE_STAT columns

Name	Type	Description
nodename	name	Node name
schemaname	name	Table namespace
relname	name	Table name

Name	Type	Description
last_vacuum	timestamp with time zone	Time when the last **VACUUM** operation is performed manually
vacuum_count	bigint	Number of times of manually performing the **VACUUM** operation
last_autovacuum	timestamp with time zone	Time when the last **VACUUM** operation is performed automatically
autovacuum_count	bigint	Number of times of automatically performing the **VACUUM** operation
last_analyze	timestamp with time zone	Time when the **ANALYZE** operation is performed (both manually and automatically)
analyze_count	bigint	Number of times of performing the **ANALYZE** operation (both manually and automatically)
last_autoanalyze	timestamp with time zone	Time when the last **ANALYZE** operation is performed automatically
autoanalyze_count	bigint	Number of times of automatically performing the **ANALYZE** operation
last_change	bigint	Time when the last modification (**INSERT**, **UPDATE**, or **DELETE**) is performed

14.3.204 PGXC_TABLE_STAT

PGXC_TABLE_STAT provides statistics of all tables of the database on all CNs and DNs in the cluster. Except the **nodename** column of the name type added in front of each row, the names, types, and sequences of other columns are the same as those in the **GS_TABLE_STAT** view.

Table 14-263 PGXC_TABLE_STAT columns

Name	Type	Description
nodename	name	Node name.
schemaname	name	Table namespace.
relname	name	Table name.

Name	Type	Description
seq_scan	bigint	Number of sequential scans. Only row-store tables are counted. For a partitioned table, the sum of the number of scans of each partition is displayed.
seq_tuple_read	bigint	Number of rows scanned in sequence. Only row-store tables are counted.
index_scan	bigint	Number of index scans. Only row-store tables are counted.
index_tuple_read	bigint	Number of rows scanned by the index. Only row-store tables are counted.
tuple_inserted	bigint	Number of rows inserted.
tuple_updated	bigint	Number of rows updated.
tuple_deleted	bigint	Number of rows deleted.
tuple_hot_updated	bigint	Number of rows with HOT updates.
live_tuples	bigint	Number of live tuples. Query the view on the CN. If **ANALYZE** is executed, the total number of live tuples in the table is displayed. Otherwise, **0** is displayed. This indicator applies only to row-store tables.
dead_tuples	bigint	Number of dead tuples. Query the view on the CN. If **ANALYZE** is executed, the total number of dead tuples in the table is displayed. Otherwise, **0** is displayed. This indicator applies only to row-store tables.

14.3.205 PGXC_THREAD_WAIT_STATUS

PGXC_THREAD_WAIT_STATUS displays all the call layer hierarchy relationship between threads of the SQL statements on all the nodes in a cluster, and the waiting status of the block for each thread, so that you can easily locate the causes of process response failures and similar phenomena.

The definitions of **PGXC_THREAD_WAIT_STATUS** view and **PG_THREAD_WAIT_STATUS** view are the same, because the essence of the **PGXC_THREAD_WAIT_STATUS** view is the query summary result of the **PG_THREAD_WAIT_STATUS** view on each node in the cluster.

Table 14-264 PGXC_THREAD_WAIT_STATUS columns

Name	Type	Description
node_name	text	Current node name

Name	Type	Description
db_name	text	Database name
thread_name	text	Thread name
query_id	bigint	Query ID. It is equivalent to **debug_query_id**.
tid	bigint	Thread ID of the current thread
lwtid	integer	Lightweight thread ID of the current thread
ptid	integer	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 14-209**.
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.

Example:

Assume you run a statement on coordinator1, and no response is returned after a long period of time. In this case, establish another connection to coordinator1 to check the thread status on it.

```
 select * from pg_thread_wait_status where query_id > 0;
  node_name   | db_name  | thread_name  | query_id |       tid       | lwtid | ptid  | tlevel | smpid |
wait_status   |   wait_event
--------------+----------+--------------+----------+-----------------+-------+-------+--------+--------
+----------------------
 coordinator1 | gaussdb | gsql         | 20971544 | 140274089064208 | 22579 |       |      0 |      0 | wait node:
datanode4 |
(1 rows)
```

Furthermore, you can view the statement working status on each node in the entire cluster. In the following example, no DNs have threads blocked, and there is a huge amount of data to be read, causing slow execution.

```
select * from pgxc_thread_wait_status where query_id=20971544;
  node_name   | db_name  | thread_name  | query_id |       tid       | lwtid | ptid  | tlevel | smpid |
wait_status    |  wait_event
--------------+----------+--------------+----------+-----------------+-------+-------+--------+--------
+----------------------
 datanode1    | gaussdb | coordinator1 | 20971544 | 139902867994384 | 22735 |       |      0 |      0 | wait
node: datanode3 |
 datanode1    | gaussdb | coordinator1 | 20971544 | 139902838634256 | 22970 | 22735 |      5 |      0 |
synchronize quit     |
 datanode1    | gaussdb | coordinator1 | 20971544 | 139902607947536 | 22972 | 22735 |      5 |      1 |
synchronize quit     |
 datanode2    | gaussdb | coordinator1 | 20971544 | 140632156796688 | 22736 |       |      0 |      0 | wait
node: datanode3 |
 datanode2    | gaussdb | coordinator1 | 20971544 | 140632030967568 | 22974 | 22736 |      5 |      0 |
synchronize quit     |
 datanode2    | gaussdb | coordinator1 | 20971544 | 140632081299216 | 22975 | 22736 |      5 |     1 |
synchronize quit     |
 datanode3    | gaussdb | coordinator1 | 20971544 | 140323627988752 | 22737 |       |      0 |     0 | wait
node: datanode3 |
 datanode3    | gaussdb | coordinator1 | 20971544 | 140323523131152 | 22976 | 22737 |      5 |     0 | net
flush data       |
 datanode3    | gaussdb | coordinator1 | 20971544 | 140323548296976 | 22978 | 22737 |      5 |     1 | net
flush data
 datanode4    | gaussdb | coordinator1 | 20971544 | 140103024375568 | 22738 |       |      0 |     0 | wait
node: datanode3
 datanode4    | gaussdb | coordinator1 | 20971544 | 140102919517968 | 22979 | 22738 |      5 |     0 |
synchronize quit     |
 datanode4    | gaussdb | coordinator1 | 20971544 | 140102969849616 | 22980 | 22738 |      5 |     1 |
synchronize quit     |
 coordinator1 | gaussdb | gsql         | 20971544 | 140274089064208 | 22579 |       |      0 |     0 | wait node:
datanode4  |
(13 rows)
```

14.3.206 PGXC_TOTAL_MEMORY_DETAIL

PGXC_TOTAL_MEMORY_DETAIL displays the memory usage in the cluster. Only the system administrator or the preset role **gs_role_read_all_stats** can access this view.

Table 14-265 PGXC_TOTAL_MEMORY_DETAIL columns

Name	Type	Description
nodename	text	Node name

Name	Type	Description
memorytype	text	Memory name, which can be set to any of the following values: ● **max_process_memory**: memory used by a GaussDB(DWS) cluster instance ● **process_used_memory**: memory used by a GaussDB(DWS) 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 column store ● **cstore_used_memory**: memory used for column store ● **max_sctpcomm_memory**: maximum memory allowed for the communication library ● **sctpcomm_used_memory**: memory used for the communication library ● **sctpcomm_peak_memory**: memory peak of the communication library ● **other_used_memory**: other used memory ● **gpu_max_dynamic_memory**: maximum GPU memory ● **gpu_dynamic_used_memory**: sum of the available GPU memory and temporary GPU memory ● **gpu_dynamic_peak_memory**: maximum memory used for GPU ● **pooler_conn_memory**: memory used for pooler connections ● **pooler_freeconn_memory**: memory used for idle pooler connections

Name	Type	Description
		● **storage_compress_memory**: memory used for column-store compression and decompression ● **udf_reserved_memory**: memory reserved for the **UDF Worker** process ● **mmap_used_memory**: memory used for **mmap**
memorymbytes	integer	Size of the used memory (MB)

14.3.207 PGXC_TOTAL_SCHEMA_INFO

PGXC_TOTAL_SCHEMA_INFO displays the schema space information of all instances in the cluster, providing visibility into the schema space usage of each instance. This view can be queried only on CNs.

Table 14-266 PGXC_TOTAL_SCHEMA_INFO columns

Name	Type	Description
schemaname	text	Schema name
schemaid	oid	Schema OID
databasename	text	Database name
databaseid	oid	Database OID
nodename	text	Instance name
nodegroup	text	Name of the node group
usedspace	bigint	Size of used space
permspace	bigint	Upper limit of the space

14.3.208 PGXC_TOTAL_SCHEMA_INFO_ANALYZE

PGXC_TOTAL_SCHEMA_INFO_ANALYZE displays the overall schema space information of the cluster, including the total cluster space, average space of instances, skew ratio, maximum space of a single instance, minimum space of a single instance, and names of the instances with the maximum space and minimum space. It provides visibility into the schema space usage of the entire cluster. This view can be queried only on CNs.

Table 14-267 PGXC_TOTAL_SCHEMA_INFO_ANALYZE columns

Name	Type	Description
schemaname	text	Schema name
databasename	text	Database name
nodegroup	text	Name of the node group
total_value	bigint	Total cluster space in the current schema
avg_value	bigint	Average space of instances in the current schema
skew_percent	integer	Skew ratio
extend_info	text	Extended information, including the maximum space of a single instance, minimum space of a single instance, and names of the instances with the maximum sapce and minimum space

14.3.209 PGXC_USER_TRANSACTION

PGXC_USER_TRANSACTION provides transaction information about users on all CNs. It is accessible only to users with system administrator rights. This view is valid only when the real-time resource monitoring function is enabled, that is, when enable_resource_track is **on**.

Table 14-268 PGXC_USER_TRANSACTION columns

Name	Type	Description
node_name	name	Node name
usename	name	Username
commit_counter	bigint	Number of the commit times
rollback_counter	bigint	Number of rollbacks
resp_min	bigint	Minimum response time
resp_max	bigint	Maximum response time
resp_avg	bigint	Average response time
resp_total	bigint	Total response time

14.3.210 PGXC_VARIABLE_INFO

PGXC_VARIABLE_INFO displays information about transaction IDs and OIDs of all nodes in a cluster.

Table 14-269 PGXC_VARIABLE_INFO columns

Name	Type	Description
node_name	text	Node name
nextOid	oid	OID generated next time for a node
nextXid	xid	Transaction ID generated next time for a node
oldestXid	xid	Oldest transaction ID for a node
xidVacLimit	xid	Critical point that triggers forcible autovacuum
oldestXidDB	oid	OID of the database that has the minimum **datafrozenxid** on a node
lastExtendCSNLogpage	integer	Number of the last extended csnlog page
startExtendCSNLogpage	integer	Number of the page from which the csnlog extending starts
nextCommitSeqNo	integer	CSN generated next time for a node
latestCompletedXid	xid	Latest transaction ID on a node after the transaction commission or rollback
startupMaxXid	xid	Last transaction ID before a node is powered off

14.3.211 PGXC_WAIT_DETAIL

PGXC_WAIT_DETAIL displays detailed information about the SQL waiting hierarchy of all nodes in a cluster. This view is supported only by clusters of version 8.1.3.200 or later.

Table 14-270 PGXC_WAIT_DETAIL columns

Name	Type	Description
level	integer	Level in the wait hierarchy. The value starts with 1 and increases by 1 when there is a wait relationship.

Name	Type	Description
lock_wait_hierarchy	text	Wait hierarchy, in the format of *Node name: Process ID->Node name:Waiting process ID->Node name:Waiting process ID*->...
node_name	text	Node name
db_name	text	Database name
thread_name	text	Thread name
query_id	bigint	ID of a query statement
tid	bigint	Thread ID of the current thread
lwtid	integer	Lightweight thread ID of the current thread
ptid	integer	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
wait_event	text	Virtual ID of the transaction holding or awaiting this lock
exec_cn	boolean	SQL execution CN
wait_node	text	Lock level
query	text	Query statement
application_name	text	Name of the application connected to the backend
backend_start	timestamp with time zone	Startup time of the backend process, that is, the time when the client connects to the server
xact_start	timestamp with time zone	Start time of the current transaction
query_start	timestamp with time zone	Start time of the active query
waiting	boolean	Waiting status
state	text	Overall state of the backend

Examples

Step 1 Connect to the CN, start a transaction, and perform the update operation.

```
begin;update td set c2=6 where c1=1;
```

Step 2 Open another window to connect to the CN, start another transaction, and perform the update operation. (Do not update the same record concurrently.)

```
begin;update td set c2=6 where c1=7;
```

In this case, the update operation is blocked.

Step 3 Open another window to connect to the CN node and create an index.

```
create index c2_key on td(c2);
```

Step 4 Run the **select * from pgxc_wait_detail;** command.

```
SELECT * FROM PGXC_WAIT_DETAIL;
level |                lock_wait_hierarchy                | node_name | db_name  | thread_name |      query_id
|        tid        | lwtid  | ptid | tlevel | sm
pid | wait_status  | wait_event | exec_cn | wait_node |                query                 | application_name |
backend_start          |           xact_st
art            |           query_start          | waiting | state
-------+---------------------------------------------------+-----------+----------+-------------+-------------------
+-----------------+--------+------+--------+---
----+--------------+------------+---------+-----------+----------------------------------------+------------------
+------------------------------+-----------------
--------------+------------------------------+---------+--------
1 | cn_5001:139870843444360                          | cn_5001   | postgres | workload    | 73183493945299462 |
139870843444360 | 578531 |      |      0 |
0 | wait node    |            | t       |           | WLM fetch collect info from data nodes | workload         |
2023-03-13 13:56:56.611486+08 | 2023-03-14 11:54
:33.562808+08 | 2023-03-13 13:57:00.262736+08 | t       | active
1 | cn_5001:139870843654544                          | cn_5001   | postgres | gsql        | 73183493945299204 |
139870843654544 | 722259 |      |      0 |
0 | wait node    |            | t       |           | update td set c2=6 where c1=1;         | gsql             | 2023-03-14
11:52:05.176588+08 | 2023-03-14 11:52
:19.054727+08 | 2023-03-14 11:53:58.114794+08 | t       | active
1 | cn_5001:139870843655296                          | cn_5001   | postgres | gsql        | 73183493945299218 |
139870843655296 | 722301 |      |      0 |
0 | wait node    |            | t       |           | update td set c2=6 where c1=7;         | gsql             | 2023-03-14
11:52:08.084265+08 | 2023-03-14 11:52
:42.978132+08 | 2023-03-14 11:53:59.459575+08 | t       | active
1 | cn_5001:139870843656424                          | cn_5001   | postgres | gsql        | 73183493945299223 |
139870843656424 | 722344 |      |      0 |
0 | acquire lock | relation   | t       |           | create index c2_key on td(c2);         | gsql             | 2023-03-14
11:52:10.967028+08 | 2023-03-14 11:52
:53.463227+08 | 2023-03-14 11:54:00.25203+08  | t       | active
2 | cn_5001:139870843656424 -> cn_5001:139870843655296 | cn_5001   | postgres | gsql        |
73183493945299218 | 139870843655296 | 722344 |      |        |
|              |            | f       |           | update td set c2=6 where c1=7;         | gsql             | 2023-03-14
11:52:08.084265+08 | 2023-03-14 11:52
:42.978132+08 | 2023-03-14 11:53:59.459575+08 | t       | active
(5 rows)
```

----End

14.3.212 PGXC_WAIT_EVENTS

PGXC_WAIT_EVENTS displays statistics on the waiting status and events of each node in the cluster. The content is the same as that displayed in **GS_WAIT_EVENTS**. This view is accessible only to users with system administrators rights.

Table 14-271 PGXC_WAIT_EVENTS columns

Name	Type	Description
nodename	name	Node name.
type	text	Event type, which can be **STATUS**, **LOCK_EVENT**, **LWLOCK_EVENT**, or **IO_EVENT**.
event	text	Event name. For details, see **PG_THREAD_WAIT_STATUS**.
wait	bigint	Number of times an event occurs. This column and all the columns below are values accumulated during process running.
failed_wait	bigint	Number of waiting failures. In the current version, this column is used only for counting timeout errors and waiting failures of locks such as **LOCK** and **LWLOCK**.
total_wait_time	bigint	Total duration of the event.
avg_wait_time	bigint	Average duration of the event.
max_wait_time	bigint	Maximum wait time of the event.
min_wait_time	bigint	Minimum wait time of the event.

14.3.213 PGXC_WLM_OPERATOR_HISTORY

PGXC_WLM_OPERATOR_HISTORY displays the operator information of completed jobs executed on all CNs. This view is used to query data from GaussDB(DWS). Data in the database is cleared every 3 minutes.

Only the system administrator or the preset role **gs_role_read_all_stats** can access this view. For details about columns in the view, see **Table 14-272**.

Table 14-272 GS_WLM_OPERATOR_INFO columns

Name	Type	Description
nodename	text	Name of the CN where the statement is executed.
queryid	bigint	Internal **query_id** used for statement execution.
pid	bigint	ID of the backend thread.
plan_node_id	integer	**plan_node_id** of the execution plan of a query.

Name	Type	Description
plan_node_name	text	Name of the operator corresponding to **plan_node_id**.
start_time	timestamp with time zone	Time when an operator starts to process the first data record.
duration	bigint	Total execution time of an operator. The unit is ms.
query_dop	integer	DOP of the current operator.
estimated_rows	bigint	Number of rows estimated by the optimizer.
tuple_processed	bigint	Number of elements returned by the current operator.
min_peak_memory	integer	Minimum peak memory used by the current operator on all DNs. The unit is MB.
max_peak_memory	integer	Maximum peak memory used by the current operator on all DNs. The unit is MB.
average_peak_memory	integer	Average peak memory used by the current operator on all DNs. The unit is MB.
memory_skew_percent	integer	Memory usage skew of the current operator among DNs.
min_spill_size	integer	Minimum spilled data among all DNs when a spill occurs. The unit is MB. Default value: **0**.
max_spill_size	integer	Maximum spilled data among all DNs when a spill occurs. The unit is MB. Default value: **0**.
average_spill_size	integer	Average spilled data among all DNs when a spill occurs. The unit is MB. Default value: **0**.
spill_skew_percent	integer	DN spill skew when a spill occurs.
min_cpu_time	bigint	Minimum execution time of the operator on all DNs. The unit is ms.
max_cpu_time	bigint	Maximum execution time of the operator on all DNs. The unit is ms.
total_cpu_time	bigint	Total execution time of the operator on all DNs. The unit is ms.
cpu_skew_percent	integer	Skew of the execution time among DNs.

Name	Type	Description
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

14.3.214 PGXC_WLM_OPERATOR_INFO

PGXC_WLM_OPERATOR_INFO displays the operator information of completed jobs executed on CNs. The data in this view is obtained from **GS_WLM_OPERATOR_INFO**.

Only the system administrator or the preset role **gs_role_read_all_stats** can access this view. For details about columns in the view, see **Table 14-273**.

Table 14-273 GS_WLM_OPERATOR_INFO columns

Name	Type	Description
nodename	text	Name of the CN where the statement is executed.
queryid	bigint	Internal **query_id** used for statement execution.
pid	bigint	ID of the backend thread.
plan_node_id	integer	**plan_node_id** of the execution plan of a query.
plan_node_name	text	Name of the operator corresponding to **plan_node_id**.
start_time	timestamp with time zone	Time when an operator starts to process the first data record.
duration	bigint	Total execution time of an operator. The unit is ms.
query_dop	integer	DOP of the current operator.
estimated_rows	bigint	Number of rows estimated by the optimizer.

Name	Type	Description
tuple_processed	bigint	Number of elements returned by the current operator.
min_peak_memory	integer	Minimum peak memory used by the current operator on all DNs. The unit is MB.
max_peak_memory	integer	Maximum peak memory used by the current operator on all DNs. The unit is MB.
average_peak_memory	integer	Average peak memory used by the current operator on all DNs. The unit is MB.
memory_skew_percent	integer	Memory usage skew of the current operator among DNs.
min_spill_size	integer	Minimum spilled data among all DNs when a spill occurs. The unit is MB. Default value: **0**.
max_spill_size	integer	Maximum spilled data among all DNs when a spill occurs. The unit is MB. Default value: **0**.
average_spill_size	integer	Average spilled data among all DNs when a spill occurs. The unit is MB. Default value: **0**.
spill_skew_percent	integer	DN spill skew when a spill occurs.
min_cpu_time	bigint	Minimum execution time of the operator on all DNs. The unit is ms.
max_cpu_time	bigint	Maximum execution time of the operator on all DNs. The unit is ms.
total_cpu_time	bigint	Total execution time of the operator on all DNs. The unit is ms.
cpu_skew_percent	integer	Skew of the execution time among DNs.
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

14.3.215 PGXC_WLM_OPERATOR_STATISTICS

PGXC_WLM_OPERATOR_STATISTICS displays the operator information of jobs being executed on CNs.

Only the system administrator or the preset role **gs_role_read_all_stats** can access this view.

Table 14-274 PGXC_WLM_OPERATOR_STATISTICS columns

Name	Type	Description
queryid	bigint	Internal **query_id** used for statement execution.
pid	bigint	ID of the backend thread.
plan_node_id	integer	**plan_node_id** of the execution plan of a query.
plan_node_name	text	Name of the operator corresponding to **plan_node_id**.
start_time	timestamp with time zone	Time when an operator starts to process the first data record.
duration	bigint	Total execution time of an operator. The unit is ms.
status	text	Execution status of the current operator. Its value can be **finished** or **running**.
query_dop	integer	DOP of the current operator.
estimated_rows	bigint	Number of rows estimated by the optimizer.
tuple_processed	bigint	Number of elements returned by the current operator.
min_peak_memory	integer	Minimum peak memory used by the current operator on all DNs. The unit is MB.
max_peak_memory	integer	Maximum peak memory used by the current operator on all DNs. The unit is MB.
average_peak_memory	integer	Average peak memory used by the current operator on all DNs. The unit is MB.
memory_skew_percent	integer	Memory usage skew of the current operator among DNs.
min_spill_size	integer	Minimum spilled data among all DNs when a spill occurs. The default value is **0**. The unit is MB.

Name	Type	Description
max_spill_size	integer	Maximum spilled data among all DNs when a spill occurs. The default value is **0**. The unit is MB.
average_spill_size	integer	Average spilled data among all DNs when a spill occurs. The default value is **0**. The unit is MB.
spill_skew_percent	integer	DN spill skew when a spill occurs.
min_cpu_time	bigint	Minimum execution time of the operator on all DNs. The unit is ms.
max_cpu_time	bigint	Maximum execution time of the operator on all DNs. The unit is ms.
total_cpu_time	bigint	Total execution time of the operator on all DNs. The unit is ms.
cpu_skew_percent	integer	Skew of the execution time among DNs.
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

14.3.216 PGXC_WLM_SESSION_INFO

PGXC_WLM_SESSION_INFO displays load management information for completed jobs executed on all CNs. The data in this view is obtained from **GS_WLM_SESSION_INFO**.

The columns are similar to those in the **GS_WLM_SESSION_HISTORY** view. For details, see **Table 14-275**.

Table 14-275 GS_WLM_SESSION_HISTORY columns

Name	Type	Description
datid	oid	OID of the database connected to the backend.

Name	Type	Description
dbname	text	Name of the database connected to the backend.
schemaname	text	Schema name.
nodename	text	Name of the CN where the statement is run.
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 only be non-null for IP connections, and only when **log_hostname** is enabled.
client_port	integer	TCP port number used by the client to communicate with the backend. If a Unix socket is used, it is **–1**.
query_band	text	Job type, which can be set through the GUC parameter **query_band** and is null string by default.
block_time	bigint	Blocking time before statement execution, including statement parsing and optimization time, in milliseconds.
start_time	timestamp with time zone	Start time of statement execution.
finish_time	timestamp with time zone	End time of statement execution.
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.

Name	Type	Description
status	text	End status of statement execution: **finished** for normal and **aborted** for abnormal. The statement status recorded here should be the database server execution status. When the server-side execution is successful and an error occurs when the result set is returned, the statement should be **finished**.
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. The unit is MB.
min_peak_memory	integer	Minimum memory peak of a statement across all DNs. The unit is MB.
max_peak_memory	integer	Maximum memory peak of a statement across all DNs. The unit is MB.
average_peak_memory	integer	Average memory usage during statement execution. The unit is MB.
memory_skew_percent	integer	Memory usage skew of a statement among DNs.
spill_info	text	Statement spill information on all DNs. **None**: The statement has not been spilled to disks on any DNs. **All**: The statement has been 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. The default value is **0**. The unit is MB.
max_spill_size	integer	Maximum spilled data among all DNs when a spill occurs. The default value is **0**. The unit is MB.
average_spill_size	integer	Average spilled data among all DNs when a spill occurs. The default value is **0**. The unit is MB.
spill_skew_percent	integer	DN spill skew when a spill occurs.

Name	Type	Description
min_dn_time	bigint	Minimum execution time of a statement across all DNs. The unit is ms.
max_dn_time	bigint	Maximum execution time of a statement across all DNs. The unit is ms.
average_dn_time	bigint	Average execution time of a statement across all DNs. The unit is ms.
dntime_skew_percent	integer	Execution time skew of a statement among DNs.
min_cpu_time	bigint	Minimum CPU time of a statement across all DNs. The unit is ms.
max_cpu_time	bigint	Maximum CPU time of a statement across all DNs. The unit is ms.
total_cpu_time	bigint	Total CPU time of a statement across all DNs. The unit is ms.
cpu_skew_percent	integer	CPU time skew of a statement among DNs.
min_peak_iops	integer	Minimum I/O peak of a statement on all DNs (times/s in column-store tables and 10,000 times/s in row-store tables). This function is not enabled in clusters of version 8.1.3. Therefore, you are not advised to refer to this column to analyze memory problems.
max_peak_iops	integer	Maximum I/O peak of a statement on all DNs (times/s in column-store tables and 10,000 times/s in row-store tables). This function is not enabled in clusters of version 8.1.3. Therefore, you are not advised to refer to this column to analyze memory problems.
average_peak_iops	integer	Average I/O peak of a statement on all DNs (times/s in column-store tables and 10,000 times/s in row-store tables). This function is not enabled in clusters of version 8.1.3. Therefore, you are not advised to refer to this column to analyze memory problems.
iops_skew_percent	integer	I/O skew of a statement among DNs. This function is not enabled in clusters of version 8.1.3. You are not advised to refer to this column to analyze memory problems.

Name	Type	Description
warning	text	Warning. The following warnings and warnings related to SQL self-diagnosis tuning 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	Executed statement.
query_plan	text	Execution plan of a statement.
node_group	text	Logical cluster of the user running the statement.
pid	bigint	PID of the backend thread for the statement.
lane	text	Fast/Slow lane where the statement is executed.
unique_sql_id	bigint	ID of the normalized unique SQL.

14.3.217 PGXC_WLM_SESSION_HISTORY

PGXC_WLM_SESSION_HISTORY displays load management information for completed jobs executed on all CNs. This view is used by Data Manager to query data from a database. Data in the database is cleared every 3 minutes. For details, see **GS_WLM_SESSION_HISTORY**.

The columns are similar to those in **GS_WLM_SESSION_HISTORY**. For details, see **Table 14-276**.

Table 14-276 GS_WLM_SESSION_HISTORY columns

Name	Type	Description
datid	oid	OID of the database connected to the backend.
dbname	text	Name of the database connected to the backend.
schemaname	text	Schema name.
nodename	text	Name of the CN where the statement is run.

Name	Type	Description
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 only be non-null for IP connections, and only when **log_hostname** is enabled.
client_port	integer	TCP port number used by the client to communicate with the backend. If a Unix socket is used, it is **–1**.
query_band	text	Job type, which can be set through the GUC parameter **query_band** and is null string by default.
block_time	bigint	Blocking time before statement execution, including statement parsing and optimization time, in milliseconds.
start_time	timestamp with time zone	Start time of statement execution.
finish_time	timestamp with time zone	End time of statement execution.
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	End status of statement execution: **finished** for normal and **aborted** for abnormal. The statement status recorded here should be the database server execution status. When the server-side execution is successful and an error occurs when the result set is returned, the statement should be **finished**.
abort_info	text	Exception information displayed if the final statement execution status is **aborted**.
resource_pool	text	Resource pool used by the user.

Name	Type	Description
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 a statement across all DNs. The unit is MB.
max_peak_memory	integer	Maximum memory peak of a statement across all DNs. The unit is MB.
average_peak_memory	integer	Average memory usage during statement execution. The unit is MB.
memory_skew_percent	integer	Memory usage skew of a statement among DNs.
spill_info	text	Statement spill information on all DNs. **None**: The statement has not been spilled to disks on any DNs. **All**: The statement has been 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. The default value is **0**. The unit is MB.
max_spill_size	integer	Maximum spilled data among all DNs when a spill occurs. The default value is **0**. The unit is MB.
average_spill_size	integer	Average spilled data among all DNs when a spill occurs. The default value is **0**. The unit is MB.
spill_skew_percent	integer	DN spill skew when a spill occurs.
min_dn_time	bigint	Minimum execution time of a statement across all DNs. The unit is ms.
max_dn_time	bigint	Maximum execution time of a statement across all DNs. The unit is ms.
average_dn_time	bigint	Average execution time of a statement across all DNs. The unit is ms.
dntime_skew_percent	integer	Execution time skew of a statement among DNs.
min_cpu_time	bigint	Minimum CPU time of a statement across all DNs. The unit is ms.

Name	Type	Description
max_cpu_time	bigint	Maximum CPU time of a statement across all DNs. The unit is ms.
total_cpu_time	bigint	Total CPU time of a statement across all DNs. The unit is ms.
cpu_skew_percent	integer	CPU time skew of a statement among DNs.
min_peak_iops	integer	Minimum I/O peak of a statement on all DNs (times/s in column-store tables and 10,000 times/s in row-store tables). This function is not enabled in clusters of version 8.1.3. Therefore, you are not advised to refer to this column to analyze memory problems.
max_peak_iops	integer	Maximum I/O peak of a statement on all DNs (times/s in column-store tables and 10,000 times/s in row-store tables). This function is not enabled in clusters of version 8.1.3. Therefore, you are not advised to refer to this column to analyze memory problems.
average_peak_iops	integer	Average I/O peak of a statement on all DNs (times/s in column-store tables and 10,000 times/s in row-store tables). This function is not enabled in clusters of version 8.1.3. Therefore, you are not advised to refer to this column to analyze memory problems.
iops_skew_percent	integer	I/O skew of a statement among DNs. This function is not enabled in clusters of version 8.1.3. You are not advised to refer to this column to analyze memory problems.
warning	text	Warning. The following warnings and warnings related to SQL self-diagnosis tuning 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	Executed statement.
query_plan	text	Execution plan of a statement.

Name	Type	Description
node_group	text	Logical cluster of the user running the statement.
pid	bigint	PID of the backend thread for the statement.
lane	text	Fast/Slow lane where the statement is executed.
unique_sql_id	bigint	ID of the normalized unique SQL.

14.3.218 PGXC_WLM_SESSION_STATISTICS

PGXC_WLM_SESSION_STATISTICS displays load management information about jobs that are being executed on CNs.

Table 14-277 PGXC_WLM_SESSION_STATISTICS columns

Name	Type	Description
datid	oid	OID of the database connected to the backend.
dbname	name	Name of the database connected to the backend.
schemaname	text	Schema name.
nodename	text	Name of the CN 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 only be non-null for IP connections, and only when **log_hostname** is enabled.
client_port	integer	TCP port number used by the client to communicate with the backend. If a Unix socket is used, it is **–1**.

Name	Type	Description
query_band	text	Job type, which can be set through the GUC parameter **query_band** and is null string by default.
pid	bigint	ID of the backend thread.
block_time	bigint	Block time before the statement is executed. The unit is ms.
start_time	timestamp with time zone	Start time of statement execution.
duration	bigint	Time that the statement has been executed. The unit is ms.
estimate_total_time	bigint	Estimated time of statement execution. The unit is ms.
estimate_left_time	bigint	Estimated remaining time of statement execution. The unit is ms.
enqueue	text	Workload management resource status.
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 a statement across all DNs. The unit is MB.
max_peak_memory	integer	Maximum memory peak of a statement across all DNs. The unit is MB.
average_peak_memory	integer	Average memory usage during statement execution. The unit is MB.
memory_skew_percent	integer	Memory usage skew of a statement among DNs.
spill_info	text	Statement spill information on all DNs. **None**: The statement has not been spilled to disks on any DNs. **All**: The statement has been 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. The unit is MB. Default value: **0**.

Name	Type	Description
max_spill_size	integer	Maximum spilled data among all DNs when a spill occurs. The unit is MB. Default value: **0**.
average_spill_size	integer	Average spilled data among all DNs when a spill occurs. The unit is MB. Default value: **0**.
spill_skew_percent	integer	DN spill skew when a spill occurs.
min_dn_time	bigint	Minimum execution time of a statement across all DNs. The unit is ms.
max_dn_time	bigint	Maximum execution time of a statement across all DNs. The unit is ms.
average_dn_time	bigint	Average execution time of a statement across all DNs. The unit is ms.
dntime_skew_percent	integer	Execution time skew of a statement among DNs.
min_cpu_time	bigint	Minimum CPU time of a statement across all DNs. The unit is ms.
max_cpu_time	bigint	Maximum CPU time of a statement across all DNs. The unit is ms.
total_cpu_time	bigint	Total CPU time of a statement across all DNs. The unit is ms.
cpu_skew_percent	integer	CPU time skew of a statement among DNs.
min_peak_iops	integer	Minimum I/O peak of a statement on all DNs (times/s in column-store tables and 10,000 times/s in row-store tables). This function is not enabled in clusters of version 8.1.3. Therefore, you are not advised to refer to this column to analyze memory problems.
max_peak_iops	integer	Maximum I/O peak of a statement on all DNs (times/s in column-store tables and 10,000 times/s in row-store tables). This function is not enabled in clusters of version 8.1.3. Therefore, you are not advised to refer to this column to analyze memory problems.
average_peak_iops	integer	Average I/O peak of a statement on all DNs (times/s in column-store tables and 10,000 times/s in row-store tables). This function is not enabled in clusters of version 8.1.3. Therefore, you are not advised to refer to this column to analyze memory problems.

Name	Type	Description
iops_skew_percent	integer	I/O skew across DNs. This function is not enabled in clusters of version 8.1.3. You are not advised to analyze memory problems by referring to this column.
warning	text	Warning. The following warnings and warnings related to SQL self-diagnosis tuning 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.
query_plan	text	Execution plan of a statement.
node_group	text	Logical cluster of the user running the statement.

14.3.219 PGXC_WLM_WORKLOAD_RECORDS

PGXC_WLM_WORKLOAD_RECORDS displays the status of job executed by the current user on CNs. Only the system administrator or the preset role **gs_role_read_all_stats** can access this view. This view is available only when **enable_dynamic_workload** is set to **on**.

Table 14-278 PGXC_WLM_WORKLOAD_RECORDS columns

Name	Type	Description
node_name	text	Name of the CN where the job is executed
thread_id	bigint	ID of the backend thread
processid	integer	lwpid of a thread
timestamp	bigint	Time when a statement starts to be executed
username	name	Name of the user logging in to the backend
memory	integer	Memory required by a statement
active_points	integer	Number of resources consumed by a statement in a resource pool

Name	Type	Description
max_points	integer	Maximum number of resources in a resource pool
priority	integer	Priority of a job
resource_pool	text	Resource pool of a job
status	text	Job execution status. Its value can be: ● **pending** ● **running** ● **finished** ● **aborted** ● **unknown**
control_group	text	Cgroups used by a job
enqueue	text	Queue that a job is in. Its value can be: ● **GLOBAL**: global queue ● **RESPOOL**: resource pool queue ● **ACTIVE**: not in a queue
query	text	Statement that is being executed

14.3.220 PGXC_WORKLOAD_SQL_COUNT

PGXC_WORKLOAD_SQL_COUNT displays statistics on the number of SQL statements executed in workload Cgroups on all CNs in a cluster, including the number of **SELECT**, **UPDATE**, **INSERT**, and **DELETE** statements and the number of DDL, DML, and DCL statements. Only the system administrator or the preset role **gs_role_read_all_stats** can access this view.

Table 14-279 PGXC_WORKLOAD_SQL_COUNT columns

Name	Type	Description
node_name	name	Node name
workload	name	Workload Cgroup name
select_count	bigint	Number of **SELECT** statements
update_count	bigint	Number of **UPDATE** statements
insert_count	bigint	Number of **INSERT** statements

Name	Type	Description
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

14.3.221 PGXC_WORKLOAD_SQL_ELAPSE_TIME

PGXC_WORKLOAD_SQL_ELAPSE_TIME displays statistics on the response time of SQL statements in workload Cgroups on all CNs in a cluster, including the maximum, minimum, average, and total response time of **SELECT**, **UPDATE**, **INSERT**, and **DELETE** statements. The unit is microsecond. Only the system administrator or the preset role **gs_role_read_all_stats** can access this view.

Table 14-280 PGXC_WORKLOAD_SQL_ELAPSE_TIME columns

Name	Type	Description
node_name	name	Node name
workload	name	Workload Cgroup name
total_select_elapse	bigint	Total response time of **SELECT** statements
max_select_elapse	bigint	Maximum response time of **SELECT** statements
min_select_elapse	bigint	Minimum response time of **SELECT** statements
avg_select_elapse	bigint	Average response time of **SELECT** statements
total_update_elapse	bigint	Total response time of **UPDATE** statements
max_update_elapse	bigint	Maximum response time of **UPDATE** statements
min_update_elapse	bigint	Minimum response time of **UPDATE** statements
avg_update_elapse	bigint	Average response time of **UPDATE** statements

Name	Type	Description
total_insert_elapse	bigint	Total response time of **INSERT** statements
max_insert_elapse	bigint	Maximum response time of **INSERT** statements
min_insert_elapse	bigint	Minimum response time of **INSERT** statements
avg_insert_elapse	bigint	Average response time of **INSERT** statements
total_delete_elapse	bigint	Total response time of **DELETE** statements
max_delete_elapse	bigint	Maximum response time of **DELETE** statements
min_delete_elapse	bigint	Minimum response time of **DELETE** statements
avg_delete_elapse	bigint	Average response time of **DELETE** statements

14.3.222 PGXC_WORKLOAD_TRANSACTION

PGXC_WORKLOAD_TRANSACTION provides transaction information about workload cgroups on all CNs. Only the system administrator or the preset role **gs_role_read_all_stats** can access this view. This view is valid only when the real-time resource monitoring function is enabled, that is, when **enable_resource_track** is **on**.

Table 14-281 PGXC_WORKLOAD_TRANSACTION columns

Name	Type	Description
node_name	name	Node name
workload	name	Workload Cgroup name
commit_counter	bigint	Number of the commit times
rollback_counter	bigint	Number of rollbacks
resp_min	bigint	Minimum response time (unit: μs)
resp_max	bigint	Maximum response time (unit: μs)

Name	Type	Description
resp_avg	bigint	Average response time (unit: μs)
resp_total	bigint	Total response time (unit: μs)

14.3.223 PLAN_TABLE

PLAN_TABLE displays the plan information collected by **EXPLAIN PLAN**. Plan information is in a session-level life cycle. After the session exits, the data will be deleted. Data is isolated between sessions and between users.

Table 14-282 PLAN_TABLE columns

Name	Type	Description
statement_id	varchar2(30)	Query tag specified by a user
plan_id	bigint	ID of a plan to be queried
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 parameters
object_name	name	Name of an operated object. It is defined by users, not the object alias used in the query.
object_type	varchar2(30)	Object type
object_owner	name	User-defined schema to which an object belongs
projection	varchar2(4000)	Returned column information

NOTE

- A valid **object_type** value consists of a relkind type defined in PG_CLASS (**TABLE**, **INDEX**, **SEQUENCE**, **VIEW**, **FOREIGN TABLE**, **COMPOSITE TYPE**, or **TOASTVALUE TOAST** table) 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.

14.3.224 PV_FILE_STAT

By collecting statistics about the data file I/Os, **PV_FILE_STAT** displays the I/O performance of the data to detect the performance problems, such as abnormal I/O operations.

Table 14-283 PV_FILE_STAT columns

Name	Type	Description
filenum	oid	File ID
dbid	oid	Database ID
spcid	oid	ID of a tablespace
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 files, in microseconds
writetim	bigint	Total duration of writing files, in microseconds
avgiotim	bigint	Average duration of reading and writing files, in microseconds
lstiotim	bigint	Duration of the last file reading, in microseconds
miniotim	bigint	Minimum duration of reading and writing files, in microseconds
maxiowtm	bigint	Maximum duration of reading and writing files, in microseconds

14.3.225 PV_INSTANCE_TIME

PV_INSTANCE_TIME collects statistics on the running time of processes and the time consumed in each execution phase, in microseconds.

PV_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 spent
- **EXECUTION_TIME**: time spent within executors
- **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**: I/O time spent

Table 14-284 PV_INSTANCE_TIME columns

Name	Type	Description
stat_id	integer	Type ID
stat_name	text	Running time type name
value	bigint	Running time value

14.3.226 PV_OS_RUN_INFO

PV_OS_RUN_INFO displays the running status of the current operating system.

Table 14-285 PV_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

14.3.227 PV_SESSION_MEMORY

PV_SESSION_MEMORY displays statistics about memory usage at the session level in the unit of MB, including all the memory allocated to Postgres and Stream threads on DNs for jobs currently executed by users.

Table 14-286 PV_SESSION_MEMORY columns

Name	Type	Description
sessid	text	Thread start time and ID
init_mem	integer	Memory allocated to the currently executed task before the task enters the executor, in MB

Name	Type	Description
used_mem	intege r	Memory allocated to the currently executed task, in MB
peak_mem	intege r	Peak memory allocated to the currently executed task, in MB

14.3.228 PV_SESSION_MEMORY_DETAIL

PV_SESSION_MEMORY_DETAIL displays statistics about thread memory usage by memory context.

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 pv_session_memctx_detail (***threadid***,'');** statement to record information about all memory contexts of a thread into the *threadid_timestamp***.log** file in the **/tmp/dumpmem** directory. *threadid* can be obtained from the following table.

Table 14-287 PV_SESSION_MEMORY_DETAIL columns

Name	Type	Description
sessid	text	Thread start time+thread ID (string: *timestamp.threadid*)
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.

Example

Query the usage of all MemoryContexts on the current node.

Locate the thread in which the MemoryContext is created and used based on **sessid**. Check whether the memory usage meets the expectation based on **totalsize**, **freesize**, and **usedsize** to see whether memory leakage may occur.

```
SELECT * FROM PV_SESSION_MEMORY_DETAIL order by totalsize desc;
         sessid          |        sesstype         |                contextname                 | level |           parent
| totalsize | freesize | usedsize
----------------------------+-------------------------+--------------------------------------------+-------
+-------------------------------+-----------+----------+----------
 0.139975915622720          | postmaster              | gs_signal                                  |     1 |
TopMemoryContext            |  17209904 |  8081136 |  9128768
 1667462258.139973631031040 | postgres                | SRF multi-call context                     |     5 |
FunctionScan_139973631031040 |   1725504 |     3168 |  1722336
 1667461280.139973666686720 | postgres                | CacheMemoryContext                         |     1 |
TopMemoryContext            |   1472544 |   284456 |  1188088
 1667450443.139973877479168 | postgres                | CacheMemoryContext                         |     1 |
TopMemoryContext            |   1472544 |   356088 |  1116456
 1667462258.139973631031040 | postgres                | CacheMemoryContext                         |     1 |
TopMemoryContext            |   1472544 |   128216 |  1344328
 1667461250.139973915236096 | postgres                | CacheMemoryContext                         |     1 |
TopMemoryContext            |   1472544 |   226352 |  1246192
 1667450439.139974010144512 | WLMarbiter              | CacheMemoryContext                         |     1 |
TopMemoryContext            |   1472544 |   386736 |  1085808
 1667450439.139974151726848 | WDRSnapshot             | CacheMemoryContext                         |     1 |
TopMemoryContext            |   1472544 |   159720 |  1312824
 1667450439.139974026925824 | WLMmonitor              | CacheMemoryContext                         |     1 |
TopMemoryContext            |   1472544 |   297976 |  1174568
 1667451036.139973746386688 | postgres                | CacheMemoryContext                         |     1 |
TopMemoryContext            |   1472544 |   208064 |  1264480
 1667461250.139973950891776 | postgres                | CacheMemoryContext                         |     1 |
TopMemoryContext            |   1472544 |   270016 |  1202528
 1667450439.139974076212992 | WLMCalSpaceInfo         | CacheMemoryContext                         |     1 |
TopMemoryContext            |   1472544 |   393952 |  1078592
 1667450439.139974092994304 | WLMCollectWorker        | CacheMemoryContext                         |     1 |
TopMemoryContext            |   1472544 |    94848 |  1377696
 1667461254.139973971343104 | postgres                | CacheMemoryContext                         |     1 |
TopMemoryContext            |   1472544 |   338544 |  1134000
 1667461280.139973822945024 | postgres                | CacheMemoryContext                         |     1 |
TopMemoryContext            |   1472544 |   284456 |  1188088
 1667450439.139974202070784 | JobScheduler            | CacheMemoryContext                         |     1 |
TopMemoryContext            |   1472544 |   216728 |  1255816
 1667450454.139973860697856 | postgres                | CacheMemoryContext                         |     1 |
TopMemoryContext            |   1472544 |   388384 |  1084160
 0.139975915622720          | postmaster              | Postmaster                                 |     1 |
TopMemoryContext            |   1004288 |    88792 |   915496
 1667450439.139974218852096 | AutoVacLauncher         | CacheMemoryContext                         |     1 |
TopMemoryContext            |    948256 |   183488 |   764768
 1667461250.139973915236096 | postgres                | TempSmallContextGroup                      |     0
|                             |    584448 |   148032 |      119
 1667462258.139973631031040 | postgres                | TempSmallContextGroup                      |     0
|                             |    579712 |   162128 |      123
```

14.3.229 PV_SESSION_STAT

PV_SESSION_STAT displays session state statistics based on session threads or the **AutoVacuum** thread.

Table 14-288 PV_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

14.3.230 PV_SESSION_TIME

PV_SESSION_TIME displays statistics about the running time of session threads and time consumed in each execution phase, in microseconds.

Table 14-289 PV_SESSION_TIME columns

Name	Type	Description
sessid	text	Thread ID and start time
stat_id	integer	Statistics ID
stat_name	text	Running time type name
value	bigint	Running time value

14.3.231 PV_TOTAL_MEMORY_DETAIL

PV_TOTAL_MEMORY_DETAIL displays statistics about memory usage of the current database node in the unit of MB.

Table 14-290 PV_TOTAL_MEMORY_DETAIL columns

Name	Type	Description
nodename	text	Node name

Name	Type	Description
memorytype	text	Memory type. Its value can be: ● **max_process_memory**: memory used by a GaussDB(DWS) cluster instance ● **process_used_memory**: memory used by a GaussDB(DWS) 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 column store ● **cstore_used_memory**: memory used for column store ● **max_sctpcomm_memory**: maximum memory allowed for the communication library ● **sctpcomm_used_memory**: memory used for the communication library ● **sctpcomm_peak_memory**: memory peak of the communication library ● **other_used_memory**: other used memory ● **gpu_max_dynamic_memory**: maximum GPU memory ● **gpu_dynamic_used_memory**: sum of the available GPU memory and temporary GPU memory ● **gpu_dynamic_peak_memory**: maximum memory used for GPU ● **pooler_conn_memory**: memory used for pooler connections ● **pooler_freeconn_memory**: memory used for idle pooler connections ● **storage_compress_memory**: memory used for column-store compression and decompression ● **udf_reserved_memory**: memory reserved for the **UDF Worker** process

Name	Type	Description
		● **mmap_used_memory**: memory used for **mmap**
memorymbytes	integer	Size of allocated memory-typed memory

14.3.232 PV_REDO_STAT

PV_REDO_STAT displays statistics on redoing Xlogs on the current node.

Table 14-291 PV_REDO_STAT columns

Name	Type	Description
phywrts	bigint	Number of physical writes
phyblkwrt	bigint	Number of physical write blocks
writetim	bigint	Time consumed by physical writes
avgiotim	bigint	Average time for each write
lstiotim	bigint	Last write time
miniotim	bigint	Minimum write time
maxiowtm	bigint	Maximum write time

14.3.233 REDACTION_COLUMNS

REDACTION_COLUMNS displays information about all redaction columns in the current database.

Table 14-292 REDACTION_COLUMNS columns

Name	Type	Description
object_owner	name	Owner of the object to be redacted.
object_name	name	Redacted object name
column_name	name	Redacted column name
function_type	integer	Redaction type
function_parameters	text	Parameter used when the redaction type is **partial** (reserved)

Name	Type	Description
regexp_pattern	text	Pattern string when the redaction type is **regexp** (reserved)
regexp_replace_string	text	Replacement string when the redaction type is **regexp** (reserved)
regexp_position	integer	Start and end replacement positions when the redaction type is **regexp** (reserved)
regexp_occurrence	integer	Replacement times when the redaction type is **regexp** (reserved)
regexp_match_parameter	text	Regular control parameter used when the redaction type is **regexp** (reserved)
function_info	text	Redaction function information
column_description	text	Description of the redacted column
inherited	bool	Whether a redacted column is inherited from another redacted column.

14.3.234 REDACTION_POLICIES

REDACTION_POLICIES displays information about all redaction objects in the current database.

Table 14-293 REDACTION_POLICIES columns

Name	Type	Description
object_owner	name	Owner of the object to be redacted.
object_name	name	Redacted object name
policy_name	name	Name of the redact policy

Name	Type	Description
expression	text	Policy effective expression (for users)
enable	boolean	Policy status (enabled or disabled)
policy_description	text	Description of a policy
inherited	bool	Whether a redaction policy is inherited from another redaction policy.

14.3.235 REMOTE_TABLE_STAT

REMOTE_TABLE_STAT provides statistics of all tables of the database on all DNs in the cluster. Except the **nodename** column of the name type added in front of each row, the names, types, and sequences of other columns are the same as those in the **GS_TABLE_STAT** view.

Table 14-294 REMOTE_TABLE_STAT columns

Name	Type	Description
nodename	name	Node name.
schemaname	name	Table namespace.
relname	name	Table name.
seq_scan	bigint	Number of sequential scans. Only row-store tables are counted. For a partitioned table, the sum of the number of scans of each partition is displayed.
seq_tuple_read	bigint	Number of rows scanned in sequence. Only row-store tables are counted.
index_scan	bigint	Number of index scans. Only row-store tables are counted.
index_tuple_read	bigint	Number of rows scanned by the index. Only row-store tables are counted.
tuple_inserted	bigint	Number of rows inserted.
tuple_updated	bigint	Number of rows updated.
tuple_deleted	bigint	Number of rows deleted.
tuple_hot_updated	bigint	Number of rows with HOT updates.

Name	Type	Description
live_tuples	bigint	Number of live tuples. Query the view on the CN. If **ANALYZE** is executed, the total number of live tuples in the table is displayed. Otherwise, **0** is displayed. This indicator applies only to row-store tables.
dead_tuples	bigint	Number of dead tuples. Query the view on the CN. If **ANALYZE** is executed, the total number of dead tuples in the table is displayed. Otherwise, **0** is displayed. This indicator applies only to row-store tables.

14.3.236 USER_COL_COMMENTS

USER_COL_COMMENTS stores the column comments of the tables and views that the current user can access.

Name	Type	Description
column_name	character varying(64)	Column name
table_name	character varying(64)	Table/View name
owner	character varying(64)	Owner of a table/view
comments	text	Comments

14.3.237 USER_CONSTRAINTS

USER_CONSTRAINTS displays the table constraint information accessible to the current user.

Name	Type	Description
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 constraint-related table

Name	Type	Description
index_owner	character varying(64)	Owner of constraint-related index (only for the unique constraint and primary key constraint)
index_name	character varying(64)	Name of constraint-related index (only for the unique constraint and primary key constraint)

Example

Query constraints on a specified table owned by the current user. Replace **t1** with the actual table name.

```
SELECT * FROM USER_CONSTRAINTS WHERE table_name='t1';
 constraint_name | constraint_type | table_name | index_owner |  index_name
-----------------+-----------------+------------+-------------+---------------
 c_custkey_key   | p               | t1         | u1          | c_custkey_key
(1 row)
```

14.3.238 USER_CONS_COLUMNS

USER_CONSTRAINTS displays the information about constraint columns of the tables accessible to the current user.

Name	Type	Description
table_name	character varying(64)	Name of constraint-related table
column_name	character varying(64)	Name of constraint-related column
constraint_name	character varying(64)	Constraint name
position	smallint	Position of the column in the table

14.3.239 USER_INDEXES

USER_INDEXES displays index information in the current schema.

Name	Type	Description
owner	character varying(64)	Owner of the index
index_name	character varying(64)	Index name
table_name	character varying(64)	Name of the table corresponding to the index

Name	Type	Description
uniqueness	text	Whether the index is a unique index
generated	character varying(1)	Whether the index name is generated by the system
partitioned	character(3)	Whether the index has the property of the partition table

14.3.240 USER_IND_COLUMNS

USER_IND_COLUMNS displays column information about all indexes accessible to the current user.

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 column in the index

14.3.241 USER_IND_EXPRESSIONS

USER_IND_EXPRESSIONS displays information about the function-based expression index accessible to the current user.

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_expression	text	The function-based index expression of a specified column

Name	Type	Description
column_position	smallint	Position of column in the index

14.3.242 USER_IND_PARTITIONS

USER_IND_PARTITIONS displays information about index partitions accessible to the current user.

Name	Type	Description
index_owner	character varying(64)	Name of the owner of the partitioned index to which the index partition belongs
schema	character varying(64)	Schema of the partitioned index to which the index partition belongs
index_name	character varying(64)	Index name of the partitioned table to which the index partition belongs
partition_name	character varying(64)	Name of the index partition
index_partition_usable	boolean	Whether the index partition is available
high_value	text	Boundary of the table partition corresponding to the index partition. For a range partition, the boundary is the upper boundary. For a list partition, the boundary is the boundary value set. Reserved field for forward compatibility. The parameter **pretty_high_value** is added in version 8.1.3 to record the information.
pretty_high_value	text	Boundary of the table partition corresponding to the index partition. For a range partition, the boundary is the upper boundary. For a list partition, the boundary is the boundary value set. The query result is the instant decompilation output of the partition boundary expression. The output of this column is more detailed than that of **high_value**. The output information can be collation and column data type.
def_tablespace_name	name	Tablespace name of the index partition

14.3.243 USER_JOBS

USER_JOBS displays all jobs owned by the user. It is accessible only to users with system administrator rights.

Table 14-295 USER_JOBS columns

Name	Type	Description
job	int4	Job ID
log_user	name not null	User name of the job creator
priv_user	name not null	User name of the job executor
dbname	name not null	Database in which the job is created
start_date	timestamp without time zone	Job start time
start_suc	text	Start time of the 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 the ongoing job execution
this suc	text	Same as **THIS_DATE**
next_date	timestamp without time zone	Schedule time of the next job execution
next suc	text	Same as **next_date**
broken	text	Task status **Y**: the system does not try to execute the task. **N**: the system attempts to execute the task.
status	char	Status of the current job. The value range is **'r', 's', 'f', 'd'**. The default value is **'s'**. The indications are as follows: ● **r**: running ● **s**: finished ● **f**: failed ● **d**: aborted

Name	Type	Description
interval	text	Time expression used to calculate the next execution time. If this parameter is set to **null**, the job will be executed once only.
failures	smallint	Number of consecutive failures.
what	text	Body of the PL/SQL blocks or anonymous clock that the job executes

14.3.244 USER_OBJECTS

USER_OBJECTS displays all database objects accessible to the current user.

Name	Type	Description
owner	name	Owner of the object
object_name	name	Object name
object_id	oid	OID of the object
object_type	name	Type of the object
namespace	oid	Namespace containing the object
created	timestamp with time zone	Object creation time
last_ddl_time	timestamp with time zone	The last time when an object was modified.

NOTICE

For details about the value ranges of **last_ddl_time** and **last_ddl_time**, see **PG_OBJECT**.

14.3.245 USER_PART_INDEXES

USER_PART_INDEXES displays information about partitioned table indexes accessible to the current user.

Name	Type	Description
index_owner	character varying(64)	Name of the owner of the partitioned table index
schema	character varying(64)	Schema of the partitioned table index

Name	Type	Description
index_name	character varying(64)	Name of the partitioned table index
table_name	character varying(64)	Name of the partitioned table to which the partitioned table index belongs
partitioning_type	text	Partition policy of the partitioned table **NOTE** Currently, only range partitioning and list partitioning are supported.
partition_count	bigint	Number of index partitions of the partitioned table index
def_tablespace_name	name	Tablespace name of the partitioned table index
partitioning_key_count	integer	Number of partition keys of the partitioned table

14.3.246 USER_PART_TABLES

USER_PART_TABLES displays information about partitioned tables accessible to the current user.

Name	Type	Description
table_owner	character varying(64)	Name of the owner of the partitioned table
schema	character varying(64)	Schema of the partitioned table
table_name	character varying(64)	Name of the partitioned table
partitioning_type	text	Partition policy of the partitioned table **NOTE** Currently, only range partitioning and list partitioning are supported.
partition_count	bigint	Number of partitions of the partitioned table
def_tablespace_name	name	Tablespace name of the partitioned table

Name	Type	Description
partitioning_key_count	integer	Number of partition keys of the partitioned table

14.3.247 USER_PROCEDURES

USER_PROCEDURES displays information about all stored procedures and functions in the current schema.

Name	Type	Description
owner	character varying(64)	Owner of the stored procedure or the function
object_name	character varying(64)	Name of the stored procedure or the function
argument_number	smallint	Number of the input parameters in the stored procedure

14.3.248 USER_SEQUENCES

USER_SEQUENCES displays sequence information in the current schema.

Name	Type	Description
sequence_owner	character varying(64)	Owner of the sequence
sequence_name	character varying(64)	Name of the sequence

14.3.249 USER_SOURCE

USER_SOURCE displays information about stored procedures or functions in this mode, and provides the columns defined by the stored procedures or the functions.

Name	Type	Description
owner	character varying(64)	Owner of the stored procedure or the function
name	character varying(64)	Name of the stored procedure or the function
text	text	Definition of the stored procedure or the function

14.3.250 USER_SYNONYMS

USER_SYNONYMS displays synonyms accessible to the current user.

Table 14-296 USER_SYNONYMS columns

Name	Type	Description
schema_name	text	Name of the schema to which the synonym belongs.
synonym_name	text	Synonym name.
table_owner	text	Owner of the associated object.
table_schema_name	text	Schema name of the associated object.
table_name	text	Name of the associated object.

14.3.251 USER_TAB_COLUMNS

USER_TAB_COLUMNS stores information about columns of the tables and views that the current user can access.

Name	Type	Description
owner	character varying(64)	Owner of a table/view
table_name	character varying(64)	Table/View name
column_name	character varying(64)	Column name
data_type	character varying(128)	Data type of the column
column_id	integer	Sequence number of the column when a table/view is created
data_length	integer	Length of the column, in bytes
comments	text	Comments
avg_col_len	numeric	Average length of a column, in bytes

Name	Type	Description
nullable	bpchar	Whether the column can be empty. For the primary key constraint and non-null constraint, the value is **n**.
data_precision	integer	Precision of the 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 **0** for other data types.
char_length	numeric	Length of a column, in characters. This parameter is valid only for the varchar, nvarchar2, bpchar, and char types.
schema	character varying(64)	Namespace that contains the table or view.
kind	text	Type of the current record. If the column belongs to a table, the value of this column is **table**. If the column belongs to a view, the value of this column is **view**.

14.3.252 USER_TAB_COMMENTS

USER_TAB_COMMENTS displays comments about all tables and views accessible to the current user.

Name	Type	Description
owner	character varying(64)	Owner of a table/view
table_name	character varying(64)	Name of the table or the view
comments	text	Comments

14.3.253 USER_TAB_PARTITIONS

USER_TAB_PARTITIONS displays all table partitions accessible to the current user. Each partition of a partitioned table accessible to the current user has a piece of record in **USER_TAB_PARTITIONS**.

Name	Type	Description
table_owner	character varying(64)	Owner of the table that contains the partition
schema	character varying(64)	Schema of the partitioned table
table_name	character varying(64)	Table name
partition_name	character varying(64)	Name of the partition
high_value	text	Upper boundary of a range partition or boundary value set of a list partition Reserved field for forward compatibility. The parameter **pretty_high_value** is added in version 8.1.3 to record the information.
pretty_high_value	text	Upper boundary of a range partition or boundary value set of a list partition The query result is the instant decompilation output of the partition boundary expression. The output of this column is more detailed than that of **high_value**. The output information can be collation and column data type.
tablespace_name	name	Name of the tablespace that contains the partition

14.3.254 USER_TABLES

USER_TABLES displays table information in the current schema.

Name	Type	Description
owner	character varying(64)	Table owner
table_name	character varying(64)	Table name
tablespace_name	character varying(64)	Name of the tablespace that contains the table
status	character varying(8)	Whether the current record is valid

Name	Type	Description
temporary	character(1)	Whether the table is a temporary table ● **Y** indicates that it is a temporary table. ● **N** indicates that it is not a temporary table.
dropped	character varying	Whether the current record is deleted ● **YES** indicates that it is deleted. ● **NO** indicates that it is not deleted.
num_rows	numeric	Estimated number of rows in the table

14.3.255 USER_TRIGGERS

USER_TRIGGERS displays the information about triggers accessible to the current user.

Name	Type	Description
trigger_name	character varying(64)	Trigger name
table_name	character varying(64)	Name of the table that defines the trigger
table_owner	character varying(64)	Owner of the table that defines the trigger

14.3.256 USER_VIEWS

USER_VIEWS displays information about all views in the current schema.

Name	Type	Description
owner	character varying(64)	Owner of the view
view_name	character varying(64)	View name

14.3.257 V$SESSION

V$SESSION displays all session information about the current session.

Table 14-297 V$SESSION columns

Name	Type	Description
sid	bigint	OID of the background process of the current activity
serial#	integer	Sequence number of the active background process, which is **0** in GaussDB(DWS).
user#	oid	OID of the user that has logged in to the background process
username	name	Name of the user that has logged in to the background process

14.3.258 V$SESSION_LONGOPS

V$SESSION_LONGOPS displays the progress of ongoing operations.

Table 14-298 V$SESSION_LONGOPS columns

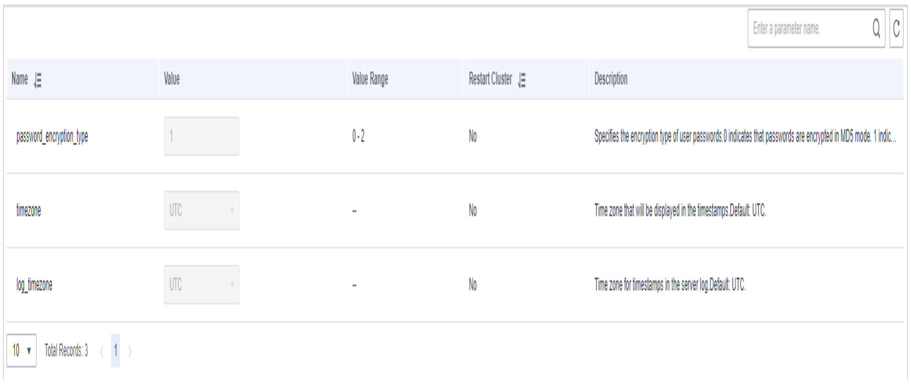
Name	Type	Description
sid	bigint	OID of the running background process
serial#	integer	Sequence number of the running background process, which is **0** in GaussDB(DWS).
sofar	integer	Completed workload, which is empty in GaussDB(DWS).
totalwork	integer	Total workload, which is empty in GaussDB(DWS).

15 GUC Parameters of the GaussDB(DWS) Database

15.1 Viewing GUC Parameters

GaussDB(DWS) GUC parameters can control database system behaviors. You can check and adjust the GUC parameters based on your business scenario and data volume.

- After a cluster is installed, you can check database parameters on the GaussDB(DWS) management console.



Name	Value	Value Range	Restart Cluster	Description
password_encryption_type	1	0 - 2	No	Specifies the encryption type of user passwords.0 indicates that passwords are encrypted in MD5 mode. 1 indic...
timezone	UTC	--	No	Time zone that will be displayed in the timestamps.Default: UTC.
log_timezone	UTC	--	No	Time zone for timestamps in the server log.Default: UTC.

- You can also connect to a cluster and run SQL commands to check the GUC parameters.
  - Run the **SHOW** command.

    NOTE

    Method 2 is limited to querying GUC parameter values of CNs. To view GUC parameter values of DNs, you can utilize Method 1 on the management console.

    To view a certain parameter, run the following command:

    ```
    SHOW server_version;
    ```

    server_version indicates the database version.

    Run the following command to view values of all parameters:

    ```
    SHOW ALL;
    ```

- Use the **pg_settings** view.

  To view a certain parameter, run the following command:

  ```
  SELECT * FROM pg_settings WHERE NAME='server_version';
  ```

  Run the following command to view values of all parameters:

  ```
  SELECT * FROM pg_settings;
  ```

15.2 Configuring GUC Parameters

To ensure the optimal performance of GaussDB(DWS), you can adjust the GUC parameters in the database.

Parameter Types and Values

- The GUC parameters of GaussDB(DWS) are classified into the following types:
  - SUSET: database administrator parameters. These parameters take effect immediately upon being set and do not require the cluster to be restarted. If a parameter of this type is set in the current session, the parameter takes effect only in the current session.
  - USERSET: common user parameters. These parameters take effect immediately upon being set and do not require the cluster to be restarted. If a parameter of this type is set in the current session, the parameter takes effect only in the current session.
  - POSTMASTER: database server parameters. Restarting the cluster is necessary to apply changes to these parameters. Once you modify a parameter of this type, a message will prompt you to restart the cluster. It is recommended to manually restart the cluster during off-peak hours for the new setting to take effect.
  - SIGHUP: global database parameters. These parameters take effect globally and cannot take effect for individual sessions.
  - BACKEND: global database parameters. These parameters take effect globally and cannot take effect for individual sessions.
- All parameter names are case insensitive. A parameter value can be an integer, floating point number, 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 system catalog **pg_settings**.
- For parameters using units, specify their units during the setting, or 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.

Setting GUC Parameters

You can configure GUC parameters in the following ways:

- Method 1: After a cluster is created, you can log in to the GaussDB(DWS) management console and modify the database parameters of the cluster.

For details, see section **Modifying Database Parameters**.

- Method 2: Connect to a cluster and run SQL commands to configure the parameters of the SUSET or USERSET type.

  Set parameters at database, user, or session levels.

  – Set a database-level parameter.
    ```
    ALTER DATABASE dbname SET paraname TO value;
    ```
    The setting takes effect in the next session.

  – Set a user-level parameter.
    ```
    ALTER USER username SET paraname TO value;
    ```
    The setting takes effect in the next session.

  – Set a session-level parameter.
    ```
    SET paraname TO value;
    ```
    Parameter value in the current session is changed. After you exit the session, the setting becomes invalid.

Procedure

The following example shows how to set **explain_perf_mode**.

Step 1 View the value of **explain_perf_mode**.

```
SHOW explain_perf_mode;
 explain_perf_mode
-------------------
 normal
(1 row)
```

Step 2 Set **explain_perf_mode**.

Perform one of the following operations:

- Set a database-level parameter.
  ```
  ALTER DATABASE gaussdb SET explain_perf_mode TO pretty;
  ```
  If the following information is displayed, the setting has been modified.
  ```
  ALTER DATABASE
  ```
  The setting takes effect in the next session.
- Set a user-level parameter.
  ```
  ALTER USER dbadmin SET explain_perf_mode TO pretty;
  ```
  If the following information is displayed, the setting has been modified.
  ```
  ALTER USER
  ```
  The setting takes effect in the next session.
- Set a session-level parameter.
  ```
  SET explain_perf_mode TO pretty;
  ```
  If the following information is displayed, the setting has been modified.
  ```
  SET
  ```

Step 3 Check whether the parameter is correctly set.

```
SHOW explain_perf_mode;
 explain_perf_mode
--------------
 pretty
(1 row)
```

----End

15.3 GUC Parameter Usage

The database provides many operation parameters. Configuration of these parameters affects the behavior of the database system. Before modifying these parameters, learn the impact of these parameters on the database. Otherwise, unexpected results may occur.

Important Notes

- 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 database.
- If the allowed maximum value of a parameter is **INT_MAX**, it indicates the maximum parameter value varies by OS.
- If the allowed maximum value of a parameter is **DBL_MAX**, it indicates the maximum parameter value varies by OS.

15.4 Connection and Authentication

15.4.1 Connection Settings

This section describes parameters related to the connection mode between the client and server.

max_connections

Parameter description: Specifies the maximum number of allowed parallel connections to the database. This parameter influences the concurrent processing capability of the cluster.

Type: POSTMASTER

Value range: an integer. For CNs, the ranges from 1 to 16384. For DNs, the value ranges from 1 to 262143. Because there are internal connections in the cluster, the maximum value is rarely reached. If **invalid value for parameter "max_connections"** is displayed in the log, you need to decrease the **max_connections** value for DNs.

Default value: **800** for CNs and **5000** for DNs. If the default value is greater than the maximum value supported by kernel (determined when the **gs_initdb** command is executed), an error message will be displayed.

Setting suggestions:

Retain the default value of this parameter on CNs. On a DN, the value of this parameter is calculated as follows:

dop_limit x 20 x 6 + 24: **dop_limit** indicates the number of CPUs of each DN in the cluster. It is calculated as follows: **dop_limit** = Number of logical CPU cores of a single server/Number of DNs of a single server.

The minimum value is 5000.

If the parameter is set to a large value, GaussDB(DWS) requires more SystemV shared memories or semaphores, which may exceed the maximum default configuration of the OS. In this case, modify the value as needed.

NOTICE

The value of **max_connections** is related to **max_prepared_transactions**. Before setting **max_connections**, ensure that the value of **max_prepared_transactions** is greater than or equal to that of **max_connections**. In this way, each session has a prepared transaction in the waiting state.

sysadmin_reserved_connections

Parameter description: Specifies the minimum number of connections reserved for administrators.

Type: POSTMASTER

Value range: an integer ranging from 0 to 262143

Default value: **3**

application_name

Parameter description: Specifies the name of the client program connecting to the database.

Type: USERSET

Value range: a string

Default value: **gsql**

connection_info

Parameter description: Specifies the database connection information, including the driver type, driver version, driver deployment path, and process owner. (This is an O&M parameter. Do not configure it by yourself.)

Type: USERSET

Value range: a string

Default value: an empty string

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 x.x.x build 39137c2d) compiled at 2022-04-01
  15:43:11 commit 3629 last mr 5138 debug","driver_path":"/usr/local/lib/
  psqlodbcw.so","os_user":"dbadmin"}
  ```
  For ODBC, JDBC, and GSQL connections, **driver_name**, **driver_version**, **driver_path**, and **os_user** are displayed by default. For other interface connections, **driver_name** and **driver_version** are displayed by default. The display of **driver_path** and **os_user** is specified by users.

15.4.2 Security and Authentication (postgresql.conf)

This section describes parameters about how to securely authenticate the client and server.

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 timeout period, the server automatically breaks the connection from the client so that the faulty client does not occupy connection resources.

Type: SIGHUP

Value range: an integer ranging from 1 to 600. The minimum unit is second (s).

Default value: **1min**

session_timeout

Parameter description: Specifies the longest duration with no operations after the connection to the server.

Type: USERSET

Value range: an integer ranging from 0 to 86400. The minimum unit is second (s). **0** means to disable the timeout.

Default value: **10 min**

NOTICE

- The gsql client of GaussDB(DWS) has an automatic reconnection mechanism. If the initialized local connection of a user to the server times out, gsql disconnects from and reconnects to the server.
- Connections from the pooler connection pool to other CNs and DNs are not controlled by the **session_timeout** parameter.

ssl_ciphers

Parameter description: Specifies the encryption algorithm list supported by the SSL.

Type: POSTMASTER

Value range: a string. Separate multiple encryption algorithms with semicolons (;).

Default value: **ALL**

NOTE

- The default value of **ssl_ciphers** is **ALL**, indicating that all the following encryption algorithms are supported. Users are advised to retain the default value, unless there are other special requirements on the encryption algorithm.
  - TLS1_3_RFC_AES_128_GCM_SHA256
  - TLS1_3_RFC_AES_256_GCM_SHA384
  - TLS1_3_RFC_CHACHA20_POLY1305_SHA256
  - TLS1_3_RFC_AES_128_CCM_SHA256
  - TLS1_3_RFC_AES_128_CCM_8_SHA256
- Currently, SSL connection authentication supports only the TLS1.3 encryption algorithm, which has better performance and security. It is also compatible with SSL connection authentication between clients that comply with TLS1.2.

ssl_renegotiation_limit

Parameter description: Specifies the traffic volume over the SSL-encrypted channel before the session key is renegotiated. The renegotiation traffic limitation 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 sent and received traffic.

Type: USERSET

NOTE

You are advised to retain the default value, that is, disable the renegotiation mechanism. You are not advised to use the **gs_guc** tool or other methods to set the **ssl_renegotiation_limit** parameter in the **postgresql.conf** file. The setting does not take effect.

Value range: an integer ranging from 0 to **INT_MAX**. The unit is KB. **0** indicates that the renegotiation mechanism is disabled.

Default value: **0**

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 a GaussDB(DWS) account.

Type: SIGHUP

NOTICE

For security purposes, do not disable the password complexity policy.

Value range: an integer, **0** or **1**

- **0** indicates that no password complexity policy is enabled.
- **1** indicates that the default password complexity policy is disabled.

Default value: **1**

password_reuse_time

Parameter description: Specifies whether to check the reuse days of the new password when you run the **ALTER USER** or **ALTER ROLE** command to change a user password.

Type: SIGHUP

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, the password can be reused if either of the following conditions is met:
- If the value of **password_reuse_time** is **0**, the days of password reuse are not limited and only the times of password reuse are limited.
- If the value of **password_reuse_max** is **0**, the times of password reuse are not limited and only the days of password reuse are limited.
- If the values of **password_reuse_time** and **password_reuse_max** are both **0**, password reuse is not limited.

Value range: a floating number ranging from 0 to 3650. The unit is day.

- **0** indicates that the password reuse days are not checked.
- A positive number indicates that the new password cannot be the one that is used within the specified days.

Default value: **60**

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.

Type: SIGHUP

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, the password can be reused if either of the following conditions is met:
- If the value of **password_reuse_time** is **0**, the days of password reuse are not limited and only the times of password reuse are limited.
- If the value of **password_reuse_max** is **0**, the times of password reuse are not limited and only the days of password reuse are limited.
- If the values of **password_reuse_time** and **password_reuse_max** are both **0**, password reuse is not limited.

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 an account is automatically unlocked.

Type: SIGHUP

NOTICE

- The locking and unlocking functions take effect only when the values of **password_lock_time** and **failed_login_attempts** are positive numbers.
- The integral part of the value of the **password_lock_time** parameter indicates the number of days and its decimal part can be converted into hours, minutes, and seconds.

Value range: a floating number ranging from 0 to 365. The unit is day.

- **0** indicates that the automatic locking function does not take effect if the password verification fails.
- A positive number indicates the duration after which an 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 in **password_lock_time**. For example, incorrect password attempts during login and password input failures when using the **ALTER USER** command

Type: SIGHUP

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 value of **failed_login_attempts**.

Default value: **10**

NOTICE

- The locking and unlocking functions take effect only when the values of **failed_login_attempts** and **password_lock_time** are positive numbers.
- **failed_login_attempts** works with the SSL connection mode of the client to identify the number of incorrect password attempts. If PGSSLMODE is set to **allow** or **prefer**, two connection requests are generated for a password connection request. One request attempts an SSL connection, and the other request attempts a non-SSL connection. In this case, the number of incorrect password attempts perceived by the user is the value of **failed_login_attempts** divided by 2.

password_encryption_type

Parameter description: Specifies the encryption type of user passwords.

Type: SIGHUP

Value range: an integer, **0**, **1**, or **2**

Table 15-1 Value description:

Value	Password Storage Format	Driver
0	Passwords are encrypted in by MD5 and stored in ciphertext.	GaussDB and open-source drivers are supported.
1	Passwords are encrypted by SHA256 and are compatible with the MD5 user authentication of the postgres client. Passwords are encrypted by MD5+SHA256.	GaussDB and open-source drivers are supported.
2	Passwords are encrypted by SHA256 and stored in ciphertext.	GaussDB drivers are supported.

NOTICE

- MD5 is not recommended because it is not a secure encryption algorithm.
- For a user created when **password_encryption_type** is set to **2**, the password has been saved using the SHA256 algorithm. In this case, changing the parameter value does not change the password storage mode in the database. Therefore, open-source clients using MD5 may still fail to connect to the database.
- When **password_encryption_type** is set to **1**, no matter the **pg_hba** authentication mode is set to **MD5** or **SHA256**, both the two encryption modes are checked to ensure function compatibility.

Default value: **1**

password_min_length

Parameter description: Specifies the minimum account password length.

Type: SIGHUP

Value range: an integer. A password can contain 6 to 999 characters.

Default value: **8**

password_max_length

Parameter description: Specifies the maximum account password length.

Type: SIGHUP

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.

Type: SIGHUP

Value range: an integer ranging from 0 to 999.

- **0** means no limit.
- A positive integer indicates the minimum number of uppercase letters in the password specified for creating an account.

Default value: **0**

password_min_lowercase

Parameter description: Specifies the minimum number of lowercase letters that an account password must contain.

Type: SIGHUP

Value range: an integer ranging from 0 to 999.

- **0** means no limit.
- A positive integer indicates the minimum number of lowercase letters in the password specified for creating an account.

Default value: **0**

password_min_digital

Parameter description: Specifies the minimum number of digits that an account password must contain.

Type: SIGHUP

Value range: an integer ranging from 0 to 999.

- **0** means no limit.
- A positive integer indicates the minimum number of digits in the password specified for creating an account.

Default value: **0**

password_min_special

Parameter description: Specifies the minimum number of special characters that an account password must contain.

Type: SIGHUP

Value range: an integer ranging from 0 to 999.

- **0** means no limit.
- A positive integer indicates the minimum number of special characters in the password specified for creating an account.

Default value: **0**

Table 15-2 Special characters

ID	Character	ID	Character	ID	Character	ID	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	,	-	-

password_effect_time

Parameter description: Specifies the validity period of an account password.

Type: SIGHUP

Value range: a floating number ranging from 0 to 999. The unit is day.

- **0** indicates the function of validity period restriction is disabled.
- A floating point number from 1 to 999 indicates the validity period of the password specified for creating an account. When the password is about to expire or has expired, the system prompts the user to change the password.

Default value: **90**

password_notify_time

Parameter description: Specifies how many days in advance users are notified before the account password expires.

Type: SIGHUP

Value range: an integer ranging from 0 to 999. The unit is day.

- **0** indicates the reminder is disabled.
- A positive integer indicates how long before expiry the reminder will appear.

Default value: **7**

15.4.3 Communication Library Parameters

This section describes parameter settings and value ranges for communication libraries.

comm_max_datanode

Parameter description: Specifies the maximum number of DNs supported by the TCP proxy communication library or SCTP communication library.

Type: USERSET

Value range: an integer ranging from 1 to 8192

Default value: actual number of DNs

NOTICE

If the number of DNs is increased, the change takes effect immediately. If the number of DNs is reduced, the cluster needs to be restarted for the change to take effect.

comm_max_stream

Parameter description: Specifies the maximum number of concurrent data streams supported by the TCP proxy communication library or SCTP communication library. The value of this parameter must be greater than: Number of concurrent data streams x Number of operators in each stream x Square of SMP.

Type: POSTMASTER

Value range: an integer ranging from 1 to 60000

Default value: calculated by the following formula: min (query_dop_limit x query_dop_limit x 2 x 20, max_process_memory (bytes) x 0.025/(Maximum number of CNs + Number of current DNs)/260. If the value is less than 1024, 1024 is used. query_dop_limit = Number of CPU cores of a single server/Number of DNs of a single server.

NOTE

- You are not advised to set this parameter to a large value because this will cause high memory usage (256 bytes x **comm_max_stream** x **comm_max_datanode**). If the number of concurrent data streams is large, the query is complex and the smp is large, resulting in insufficient memory.
- If the value of **comm_max_datanode** is small, the process memory is sufficient. In this case, you can increase the value of **comm_max_stream**.

max_stream_pool

Parameter description: Specifies the maximum number of stream threads that can be contained in a stream thread pool. This feature is supported in 8.1.2 or later.

Type: SUSET

Value range: an integer ranging from -1 to INT_MAX. The values **-1** and **0** indicate that the stream thread pool is disabled.

Default value: **65535**

NOTE

- The number of stream threads in a thread pool can be reduced in real time. If the value of this parameter is increased, the number of stream threads is increased to meet the service requirements.
- Generally, you are advised not to change the value of this parameter because the stream thread pool supports the automatic cleanup function.
- If too many idle stream threads occupy the memory, you can decrease the value of this parameter to save the memory.

comm_max_receiver

Parameter description: Specifies the maximum number of receiving threads for the TCP proxy communication library or SCTP communication library.

Type: POSTMASTER

Value range: an integer ranging from 1 to 50

Default value: 4

comm_quota_size

Parameter description: Specifies the maximum size of packets that can be consecutively sent by the TCP proxy communication library or SCTP communication library. When you use a 1GE NIC, a small value ranging from 20 KB to 40 KB is recommended.

Type: USERSET

Value range: an integer ranging from 0 to 102400. The default unit is KB. The value **0** indicates that the quota mechanism is not used.

Default value: **1 MB**

comm_usable_memory

Parameter description: Specifies the maximum memory available for buffering on the TCP proxy communication library or SCTP communication library on a single DN.

Type: POSTMASTER

Value range: an integer ranging from 1 to 256. The default unit is KB. The minimum size cannot be less than 1 GB for installation.

Default value: **max_process_memory/8**

NOTICE

This parameter must be specifically set based on environment memory and the deployment method. If it is too large, there may be out-of-memory (OOM). If it is too small, the performance of the TCP proxy communication library or SCTP communication library may deteriorate.

comm_memory_pool_percent

Parameter description: Specifies the percentage of the memory pool resources that can be used by the TCP proxy communication library or the SCTP communication library in a DN. This parameter is used to adaptively reserve memory used by the communication libraries.

Type: POSTMASTER

Value range: an integer ranging from 0 to 100

Default value: **0**

NOTICE

If the memory used by the communication library is small, set this parameter to a small value. Otherwise, set it to a large value.

comm_client_bind

Parameter description: Specifies whether to bind the client of the communication library to a specified IP address when the client initiates a connection.

Type: USERSET

Value range: Boolean

- **on** indicates that the client is bound to a specified IP address.
- **off** indicates that the client is not bound to any IP addresses.

NOTICE

If multiple IP addresses of a node in a cluster are on the same communication network segment, set this parameter to **on**. In this case, the client is bound to the IP address specified by **listen_addresses**. The concurrency performance of a cluster depends on the number of random ports because a port can be used only by one client at a time.

Default value: **off**

comm_no_delay

Parameter description: Specifies whether to use the **NO_DELAY** attribute of the communication library connection. Restart the cluster for the setting to take effect.

Type: USERSET

Value range: Boolean

Default value: off

NOTICE

If packet loss occurs because a large number of packets are received per second, set this parameter to **off** to reduce the total number of packets.

comm_debug_mode

Parameter description: Specifies the debug mode of the TCP proxy communication library or SCTP communication library, that is, whether to print logs about the communication layer. The setting is effective at the session layer.

NOTICE

When the switch is set to **on**, the number of printed logs is huge, adding extra overhead and reducing database performance. Therefore, set the switch to **on** only in the debug mode.

Type: USERSET

Value range: Boolean

- **on** indicates the detailed debug log of the communication library is printed.
- **off** indicates the detailed debug log of the communication library is not printed.

Default value: **off**

comm_ackchk_time

Parameter description: Specifies the duration after which the communication library server automatically triggers ACK when no data package is received.

Type: USERSET

Value range: an integer ranging from 0 to 20000. The unit is millisecond (ms). **0** indicates that automatic ACK triggering is disabled.

Default value: **2000**

comm_timer_mode

Parameter description: Specifies the timer mode of the TCP proxy communication library or SCTP communication library, that is, whether to print timer logs in each phase of the communication layer. The setting is effective at the session layer.

NOTICE

When the switch is set to **on**, the number of printed logs is huge, adding extra overhead and reducing database performance. Therefore, set the switch to **on** only in the debug mode.

Type: USERSET

Value range: Boolean

- **on** indicates the detailed timer log of the communication library is printed.
- **off** indicates the detailed timer log of the communication library is not printed.

Default value: **off**

comm_stat_mode

Parameter description: Specifies the statistics mode of the TCP proxy communication library or SCTP communication library, that is, whether to print statistics about the communication layer. The setting is effective at the session layer.

NOTICE

When the switch is set to **on**, the number of printed logs is huge, adding extra overhead and reducing database performance. Therefore, set the switch to **on** only in the debug mode.

Type: USERSET

Value range: Boolean

- **on** indicates the statistics log of the communication library is printed.
- **off** indicates the statistics log of the communication library is not printed.

Default value: off

enable_stateless_pooler_reuse

Parameter description: Specifies whether to enable the pooler reuse mode. The setting takes effect after the cluster is restarted.

Type: POSTMASTER

Value range: Boolean

- **on** or **true** indicates that the pooler reuse mode is enabled.
- **off** or **false** indicates that the pooler reuse mode is disabled.

NOTICE

Set this parameter to the same value for CNs and DNs. If **enable_stateless_pooler_reuse** is set to **off** for CNs and set to **on** for DNs, the cluster communication fails. Restart the cluster to make the setting take effect.

Default value: **off**

comm_cn_dn_logic_conn

Parameter description: Specifies a switch for logical connections between CNs and DNs. The parameter setting takes effect only after the cluster is restarted.

Type: POSTMASTER

Value range: Boolean

- **on** or **true** indicates that the connections between CNs and DNs are logical, with the libcomm component in use.
- **off** or **false** indicates that the connections between CNs and DNs are physical, with the libpq component in use.

NOTICE

If **comm_cn_dn_logic_conn** is set to **off** for CNs and set to **on** for DNs, cluster communication will fail. You are advised to set this parameter to the same value for all CNs and DNs. Restart the cluster to make the setting take effect.

Default value: off

15.5 Resource Consumption

15.5.1 Memory

This section describes memory parameters.

NOTICE

Parameters described in this section take effect only after the database service restarts.

enable_memory_limit

Parameter description: Specifies whether to enable the logical memory management module.

Type: POSTMASTER

Value range: Boolean

- **on** indicates the logic memory management module is enabled.
- **off** indicates the logic memory management module is disabled.

Default value: **on**

NOTICE

- If the value of max_process_memory-max_shared_memory-cstore buffers is less than 2 GB, forcibly set enable_memory_limit to off.
- The max_shared_memory parameter is closely related to the shared_buffer, max_connections, and max_prepared_transactions parameters. If the value of max_shared_memory is too large, you can decrease the values of the three parameters.
- The dynamic load management function depends on the memory management function. After the **enable_memory_limit** parameter is disabled, the dynamic load management and TopSQL functions become invalid.

max_process_memory

Parameter description: Specifies the maximum physical memory of a database node.

Type: POSTMASTER

Value range: an integer ranging from 2 x 1024 x 1024 to INT_MAX/2. The unit is KB.

Default value: The value is automatically adapted on non-secondary DNs. The formula is (Physical memory size) x 0.8/ (1 + Number of primary DNs). If the result is less than 2 GB, 2 GB is used by default. The default size of the secondary DN is 12 GB.

Setting suggestions:

On DNs, the value of this parameter is determined based on the physical system memory and the number of DNs deployed on a single node. Parameter value = (Physical memory – **vm.min_free_kbytes**) x 0.8/(n + Number of primary DNs). This parameter aims to ensure system reliability, preventing node OOM caused by increasing memory usage. **vm.min_free_kbytes** indicates OS memory reserved for kernels to receive and send data. Its value is at least 5% of the total memory. That is, **max_process_memory** = Physical memory x 0.8/ (n + Number of primary DNs). If the cluster scale (number of nodes in the cluster) is smaller than 256, n=1; if the cluster scale is larger than 256 and smaller than 512, n=2; if the cluster scale is larger than 512, n=3.

Set this parameter on CNs to the same value as that on DNs.

RAM is the maximum memory allocated to the cluster.

shared_buffers

Parameter description: Specifies the size of shared memory used by GaussDB(DWS). If this parameter is set to a large value, GaussDB(DWS) may require more System V shared memory than the default setting.

Type: POSTMASTER

Value range: an integer ranging from 128 to INT_MAX. The unit is 8 KB.

Changing the value of **BLCKSZ** will result in a change in the minimum value of the **shared_buffers**.

Default value: The value of CN is half of the value of DN. The value of DN is calculated using the following formula: **POWER(2,ROUND(LOG(2, max_process_memory/18),0))**. If the maximum value allowed by the OS is smaller than 32 MB, this parameter will be automatically changed to the maximum value allowed by the OS during database initialization.

Setting suggestions:

You are advised to set this parameter for DNs to a value greater than that for CNs, because GaussDB(DWS) pushes its most queries down to DNs.

It is recommended that **shared_buffers** be set 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 DNs on the single server) x 0.4 x 0.25

If you want to increase the value of **shared_buffers**, you also need to increase the value of **checkpoint_segments**, because a longer period of time is required to write a large amount of new or changed data.

bulk_write_ring_size

Parameter description: Specifies the size of the ring buffer used for data parallel import.

Type: USERSET

Value range: an integer ranging from 16384 to INT_MAX. The unit is KB.

Default value: **2 GB**

Setting suggestions: Increase the value of this parameter on DNs if a huge amount of data is to be imported.

buffer_ring_ratio

Parameter description: ring buffer threshold for parallel data export

Type: USERSET

Value range: integer in the range 1–1000

Default value: 250

NOTE

- The default value indicates that the threshold is 250/1000 (a quarter) of **shared_buffers**.
- The minimum value is 1/1000 of the value of **shared_buffers**.
- The maximum value is the value of **shared_buffers**.

Setting suggestions: If the cache hit ratio is not as expected during export, you are advised to configure this parameter on DNs.

temp_buffers

Parameter description: Specifies the maximum size of local temporary buffers used by each database session.

Type: USERSET

Value range: an integer ranging from 800 to INT_MAX/2. The unit is 8 KB.

Default value: **8 MB**

NOTE

- This parameter 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.
- Based on the value of **temp_buffers**, a session allocates temporary buffers as required. The cost of setting a large value in sessions that do not require many temporary buffers is only a buffer descriptor. If a buffer is used, 8192 bytes will be consumed for it.

max_prepared_transactions

Parameter description: Specifies the maximum number of transactions that can stay in the **prepared** state simultaneously. If this parameter is set to a large value, GaussDB(DWS) may require more System V shared memory than the default setting.

When GaussDB(DWS) is deployed as an HA system, set this parameter on the standby server to the same value or a value greater than that on the primary server. Otherwise, queries will fail on the standby server.

Type: POSTMASTER

Value range: an integer ranging from 0 to 536870911. The value of CN set to **0** indicates that the prepared transaction feature is disabled.

Default value: **800** for both CNs and DNs

NOTE

Set this parameter to a value greater than or equal to that of **max_connections** to avoid failures in preparation.

work_mem

Parameter description: Specifies the memory capacity to be used by internal sort operations and Hash tables before writing to temporary disk files. Sort operations are used for **ORDER BY**, **DISTINCT**, and merge joins. Hash tables are required for Hash joins as well as Hash-based aggregations and **IN** subqueries.

For a complex query, several sort or Hash operations may be running in parallel; each operation will be allowed to use as much memory as this value specifies. If the memory is insufficient, data is 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**.

Type: USERSET

Value range: an integer ranging from 64 to INT_MAX. The unit is KB.

Default value: 512 MB for small-scale memory and 2 GB for large-scale memory (If **max_process_memory** is greater than or equal to 30 GB, it is large-scale memory. Otherwise, it is small-scale memory.)

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.

- In complex serial query scenarios, each query requires five to ten associated operations. Set **work_mem** using the following formula: **work_mem** = 50% of the memory/10.
- In simple serial query scenarios, 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, use the formula: **work_mem** = **work_mem** in serialized scenario/Number of concurrent SQL statements.

query_mem

Parameter description: Specifies the memory used by query. If the value of **query_mem** is greater than 0, the optimizer adjusts the estimated query memory to this value when generating an execution plan.

Type: USERSET

Value range: **0** or an integer greater than 32 MB. The default unit is KB. If the value is set to a negative value or less than 32 MB, the default value **0** is used. In this case, the optimizer does not adjust the estimated query memory.

Default value: **0**

query_max_mem

Parameter description: Specifies the maximum memory that can be used by query. If the value of **query_max_mem** is greater than 0, when generating an execution plan, the optimizer uses this value to set the available memory for operators. If job memory usage exceeds the value of this parameter, an error is reported and the job exits.

Type: USERSET

Value range: **0** or an integer greater than 32 MB. The default unit is KB. If the value is less than 32 MB, the system automatically sets this parameter to the default value **0**. In this case, the optimizer does not limit the memory usage of jobs.

Default value: **0**

agg_max_mem

Parameter description: Specifies the maximum memory that can be used by the Agg operator when the number of aggregation columns exceeds 5. This parameter takes effect only when the value of **query_max_mem** is greater than 0. (This parameter is supported only in 8.1.3.200 and later cluster versions.)

Type: USERSET

Value range: **0** or an integer greater than 32 MB. The default unit is KB. If the value is less than 32 MB, the system automatically sets this parameter to the default value **0**. In this case, the memory usage of the Agg operator is not limited based on the value.

Default value:

- If the current cluster is upgraded from an earlier version to 8.1.3, the value in the earlier version is inherited. The default value is **INT_MAX**.
- If the current cluster version is 8.1.3, the default value is **2GB**.

maintenance_work_mem

Parameter description: Specifies the maximum size of memory to be used for 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.

Type: USERSET

Value range: an integer ranging from 1024 to INT_MAX. The unit is KB.

Default value: 512 MB for small-scale memory and 2 GB for large-scale memory (If **max_process_memory** is greater than or equal to 30 GB, it is large-scale memory. Otherwise, it is small-scale memory.)

Setting suggestions:

- You are advised to set this parameter to the same value of **work_mem** so that database dump can be cleared or restored more quickly. In a database session, only one maintenance operation can be performed at a time. Maintenance is usually performed when there are not much sessions.
- When the **Automatic Cleanup** process is running, up to **autovacuum_max_workers** times of this memory may be allocated. Set **maintenance_work_mem** to a value equal to or larger than the value of **work_mem**.
- If a large amount of data needs to be processed in the cluster, increase the value of this parameter in sessions.

psort_work_mem

Parameter description: Specifies the memory used for internal sort operations on column-store tables before they are written into 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.

Type: USERSET

NOTICE

Multiple running sessions may perform partial sorting on a table at the same time. Therefore, the total memory usage may be several times of the **psort_work_mem** value.

Value range: an integer ranging from 64 to INT_MAX. The unit is KB.

Default value: **512 MB**

max_loaded_cudesc

Parameter description: Specifies the number of loaded CuDescs per column when a column-store table is scanned. Increasing the value will improve the query performance and increase the memory usage, particularly when there are many columns in the column tables.

Type: USERSET

Value range: an integer ranging from 100 to INT_MAX/2

Default value: **1024**

NOTICE

When the value of **max_loaded_cudesc** is set to a large value, the memory may be insufficient.

max_stack_depth

Parameter description: Specifies the maximum safe depth of GaussDB(DWS) 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.

Type: SUSET

Take the following into consideration when setting this parameter:

- The ideal value of this parameter is the maximum stack size enforced by the kernel (value of **ulimit -s**).
- Setting this parameter to a value larger than the actual kernel limit means that a running recursive function may crash an individual backend process. In an OS where GaussDB(DWS) can check the kernel limit, such as the SLES, GaussDB(DWS) will prevent this parameter from being set to a value greater than the kernel limit.
- Since not all the OSs provide this function, you are advised to set a specific value for this parameter.

Value range: an integer ranging from 100 to INT_MAX. The unit is KB.

Default value: **2 MB**

NOTE

2 MB is a small value and will not incur system breakdown in general, but may lead to execution failures of complex functions.

cstore_buffers

Parameter description: Specifies the size of the shared buffer used by ORC, Parquet, or CarbonData data of column-store tables and OBS or HDFS column-store foreign tables.

Type: POSTMASTER

Value range: an integer ranging from 16384 to INT_MAX. The unit is KB.

Default value: The CN size is 32 MB, and the DN size is calculated as follows: **POWER(2,ROUND(LOG(2, max_process_memory/18),0))**.

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**.

Use **cstore_buffers** to specify the cache of ORC, Parquet, or CarbonData metadata and data for OBS or HDFS foreign tables. The metadata cache size should be 1/4 of **cstore_buffers** and not exceed 2 GB. The remaining cache is shared by column-store data and foreign table column-store data.

enable_orc_cache

Parameter description: Specifies whether to reserve 1/4 of **cstore_buffers** for storing ORC metadata when the cstore buffer is initialized.

Type: POSTMASTER

Value range: Boolean

Default value:

- **on** indicates that the orc metadata cache is enabled, which improves the query performance of the HDFS table but occupies the column-store buffer resources. The column-store performance deteriorates.
- **off** indicates the orc metadata cache is disabled.

schedule_splits_threshold

Parameter description: Specifies the maximum number of files that can be stored in memory when you schedule an HDFS foreign table. If the number is exceeded, all files in the list will be spilled to disk for scheduling.

Type: USERSET

Value range: an integer ranging from 1 to INT_MAX

Default value: **60000**

bulk_read_ring_size

Parameter description: Specifies the ring buffer size used for data parallel export.

Type: USERSET

Value range: an integer ranging from 256 to INT_MAX. The unit is KB.

Default value: **16 MB**

check_cu_size_threshold

Parameter description: If the amount of data inserted to a CU is greater than the value of this parameter when data is inserted to a column-store table, the system starts row-level size verification to prevent the generation of a CU whose size is greater than 1 GB (non-compressed size).

Type: USERSET

Value range: an integer ranging from 0 to 1048576. The unit is KB.

Default value: 1 GB

15.5.2 Statement Disk Space Control

This section describes parameters related to statement disk space control, which are used to limit the disk space usage of statements.

sql_use_spacelimit

Parameter description: Specifies the space size for files to be spilled to disks when a single SQL statement is executed on a single DN. The managed space includes the space occupied by ordinary tables, temporary tables, and intermediate result sets to be flushed to disks. System administrators are also restricted by this parameter.

Type: USERSET

Value range: an integer ranging from -1 to INT_MAX. The unit is KB. **–1** indicates no limit.

Default value: Set **sql_use_spacelimit** to 10% of the total space of the disk where the instance is.

NOTE

For example, if **sql_use_spacelimit** is set to **100** in the statement and the amount data spilled to disks on a single DN exceeds 100 KB, DWS stops the query and displays a message of threshold exceeding.

```
insert into user1.t1 select * from user2.t1;
ERROR:  The space used on DN (104 kB) has exceeded the sql use space limit (100 kB).
```

Handling suggestion:

- Optimize the statement to reduce the data spilled to disks.
- If the disk space is sufficient, increase the value of this parameter.

temp_file_limit

Parameter description: Specifies the total space for files spilled to disks in a single thread. For example, temporary files used by sorting or hash tables, or cursors in a session.

This is a session-level setting.

Type: SUSET

Value range: an integer ranging from -1 to INT_MAX. The unit is KB. **–1** indicates no limit.

Default value: Set **temp_file_limit** to 10% of the total disk space of the instance.

NOTICE

This parameter does not apply to disk space occupied by temporary tablespaces used for executing SQL queries.

bi_page_reuse_factor

Parameter description: Specifies the percentage of idle space of old pages that can be reused when page replication is used for data synchronization between primary and standby DNs in the scenario where data is inserted into row-store tables in batches.

Type: USERSET

Value range: an integer ranging from 0 to 100. The value is a percentage. Value **0** indicates that the old pages are not reused and new pages are requested.

Default value: **70**

NOTICE

- You are not advised to set this parameter to a value less than **50** (except **0**). If the idle space of the reused page is small, too much old page data will be transmitted between the primary and standby DNs. As a result, the batch insertion performance deteriorates.
- You are not advised to set this parameter to a value greater than **90**. If this parameter is set to a value greater than **90**, idle pages will be frequently queried, but old pages cannot be reused.

15.5.3 Kernel Resources

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.

But on some platforms, especially on most BSD systems, the kernel allows independent processes to open far more files than the system can really support. If the message "Too many open files" is displayed, try to reduce the setting. Generally, the number of file descriptors must be greater than or equal to the maximum number of concurrent tasks multiplied by the number of primary DNs on the current physical machine (*max_files_per_process*3).

Type: POSTMASTER

Value range: an integer ranging from 25 to INT_MAX

Default value: **1000**

max_files_per_node

Parameter description: Specifies the maximum number of files that can be opened by a single SQL statement on a single node. Generally, you do not need to set this parameter. This parameter is supported by version 8.1.3 or later clusters.

Parameter type: SUSET

Value range: an integer ranging from **–1** to **INT_MAX**. The value **–1** indicates that the maximum number is not limited.

Default value: **–1**

NOTE

- The default value of this parameter is **–1** in a new cluster. In an upgrade scenario, the default value of this parameter is retained for forward compatibility.
- If error message "The last file name is [%s] and %d files have already been opened on data node [%s] with a maximum of %d files." is displayed during statement execution, increase the value of **max_files_per_node**.

15.5.4 Cost-based Vacuum Delay

The purpose of cost-based vacuum delay is to allow administrators to reduce the I/O impact of **VACUUM** and **ANALYZE** statements on concurrently active databases. For example, when maintenance statements such as **VACUUM** and **ANALYZE** do not need to be executed quickly and do not interfere with other database operations, administrators can use this function to achieve this purpose.

NOTICE

Certain operations hold critical locks and should be complete as quickly as possible. In GaussDB(DWS), cost-based vacuum delays do not take effect during such operations. To avoid uselessly long delays in such cases, the actual delay is calculated as follows and is the maximum value of the following calculation results:

- vacuum_cost_delay*accumulated_balance/vacuum_cost_limit
- vacuum_cost_delay*4

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 value other than 0.

vacuum_cost_delay

Parameter description: Specifies the length of time that the process will sleep when **vacuum_cost_limit** has been exceeded.

Type: USERSET

Value range: an integer ranging from 0 to 100. The unit is millisecond (ms). A positive number enables cost-based vacuum delay and **0** disables cost-based vacuum delay.

Default value: **0**

NOTICE

- On many systems, the effective resolution of sleep length is 10 ms. 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 set to a small value, such as 10 or 20 milliseconds.

vacuum_cost_limit

Parameter description: Specifies the cost limit. The cleanup process will sleep if this limit is exceeded.

Type: USERSET

Value range: an integer ranging from 1 to 10000. The unit is ms.

Default value: 200

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 page.

Type: USERSET

Value range: an integer ranging from 0 to 10000. The unit is millisecond (ms).

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.

Type: USERSET

Value range: an integer ranging from 0 to 10000. The unit is millisecond (ms).

Default value: **2**

vacuum_cost_page_dirty

Parameter description: Specifies the estimated cost charged when vacuum modifies a block that was previously clean. It represents the I/Os required to flush the dirty block out to disk again.

Type: USERSET

Value range: an integer ranging from 0 to 10000. The unit is millisecond (ms).

Default value: **20**

15.5.5 Asynchronous I/O Operations

enable_adio_debug

Parameter description: Specifies whether O&M 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.

Type: SUSET

Value range: Boolean

- **on** or **true** indicates the log switch is enabled.
- **off** or **false** indicates the log switch is disabled.

Default value: **off**

enable_fast_allocate

Parameter description: Specifies whether the quick allocation switch of the disk space is enabled. This switch can be enabled only in the XFS file system.

Type: SUSET

Value range: Boolean

- **on** or **true** indicates that this function is enabled.
- **off** or **false** indicates that the function is disabled.

Default value: **off**

prefetch_quantity

Parameter description: Specifies the number of row-store prefetches using the ADIO.

Type: USERSET

Value range: an integer ranging from 1024 to 1048576. The unit is 8 KB.

Default value: **32 MB**

backwrite_quantity

Parameter description: Specifies the number of row-store writes using the ADIO.

Type: USERSET

Value range: an integer ranging from 1024 to 1048576. The unit is 8 KB.

Default value: **8MB**

cstore_prefetch_quantity

Parameter description: Specifies the number of column-store prefetches using the ADIO.

Type: USERSET

Value range: an integer. The value range is from 1024 to 1048576 and the unit is KB.

Default value: **32 MB**

cstore_backwrite_quantity

Parameter description: Specifies the number of column-store writes using the ADIO.

Type: USERSET

Value range: an integer. The value range is from 1024 to 1048576 and the unit is KB.

Default value: **8MB**

cstore_backwrite_max_threshold

Parameter description: Specifies the maximum number of column-store writes buffered in the database using the ADIO.

Type: USERSET

Value range: An integer. The value range is from 4096 to INT_MAX/2 and 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.

Type: SUSET

Value range: an integer. The value range is from 1024 to 1048576 and the unit is KB.

Default value: **8MB**

effective_io_concurrency

Parameter description: Specifies the number of requests that can be simultaneously processed by the disk subsystem. For the RAID array, the parameter value must be the number of disk drive spindles in the array.

Type: USERSET

Value range: an integer ranging from 0 to 1000

Default value: **1**

15.6 Parallel Data Import

GaussDB(DWS) provides a parallel data import function that enables a large amount of data to be imported in a fast and efficient manner. This section describes parameters for importing data in parallel in GaussDB(DWS).

raise_errors_if_no_files

Parameter description: Specifies whether distinguish between the problems "the number of imported file records is empty" and "the imported file does not exist". If this parameter is set to **true** and the problem "the imported file does not exist" occurs, GaussDB(DWS) will report the error message "file does not exist".

Type: SUSET

Value range: Boolean

- **on** indicates the messages of "the number of imported file records is empty" and "the imported file does not exist" are distinguished when files are imported.
- **off** indicates the messages of "the number of imported file records is empty" and "the imported file does not exist" are not distinguished when files are imported.

Default value: **off**

partition_mem_batch

Parameter description: To optimize the inserting of column-store partitioned tables in batches, data is cached during the inserting process and then written to the disk in batches. You can use **partition_mem_batch** to specify the number of buffers. If the value is too large, much memory will be consumed. If it is too small, the performance of inserting column-store partitioned tables in batches will deteriorate.

Type: USERSET

Value range: 1 to 65535

Default value: **256**

partition_max_cache_size

Parameter description: To optimize the inserting of column-store partitioned tables in batches, data is cached during the inserting process and then written to the disk in batches. You can use **partition_max_cache_size** to specify the size of the data buffer. If the value is too large, much memory will be consumed. If it is too small, the performance of inserting column-store partitioned tables in batches will deteriorate.

Type: USERSET

Value range: 4096 to INT_MAX/2. The minimum unit is KB.

Default value: **2 GB**

gds_debug_mod

Parameter description: Specifies whether to enable the debug function of Gauss Data Service (GDS). This parameter is used to better locate and analyze GDS faults. After the debug function is enabled, types of packets received or sent by GDS, peer end of GDS during command interaction, and other interaction information about GDS are written into the logs of corresponding nodes. In this way, state switching on the GaussDB state machine and the current state are recorded. If this function is enabled, additional log I/O resources will be consumed, affecting log performance and validity. You are advised to enable this function only when locating GDS faults.

Type: USERSET

Value range: Boolean

- **on** indicates that the GDS debug function is enabled.
- **off** indicates that the GDS debug function is disabled.

Default value: **off**

15.7 Write Ahead Logs

15.7.1 Settings

wal_level

Parameter description: Specifies the level of the information that is written to WALs.

Type: POSTMASTER

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 only contain basic information required for the recovery from a database server crash or an emergency shutdown. Archived WALs cannot be used to restore data.
- 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 is little measurable difference in performance between using **hot_standby** and **archive** levels, so feedback is welcome if any production performance impacts are noticeable.

Default value: **hot_standby**

NOTICE

- To enable WAL archiving and data streaming replication between primary and standby servers, set this parameter to **archive** or **hot_standby**.
- If this parameter is set to **archive**, **hot_standby** must be set to **off**. Otherwise, the database startup fails.

synchronous_commit

Parameter description: Specifies the synchronization mode of the current transaction.

Type: USERSET

Value range: enumerated values

- **on** indicates synchronization logs of a standby server are flushed to disks.
- **off** indicates asynchronous commit.
- **local** indicates local commit.
- **remote_write** indicates synchronization logs of a standby server are written to disks.
- **remote_receive** indicates synchronization logs of a standby server are required to receive data.

Default value: **on**

wal_buffers

Parameter description: Specifies the number of XLOG_BLCKSZs used for storing WAL data. The size of each XLOG_BLCKSZ is 8 KB.

Type: POSTMASTER

Value range: -1 to 2^{18}. The unit is 8 KB.

- If this parameter is set to **-1**, the value of **wal_buffers** is automatically changed to 1/32 of **shared_buffers**. The minimum value is 8 x **XLOG_BLCKSZ**, and the maximum value is 2048 x **XLOG_BLCKSZ**.
- If it is set to a value smaller than **8**, the value **8** is used. If it is set to a value greater than 2048, the value **2048** is used.

Default value: **256 MB**

Setting suggestions: The content of WAL buffers is written to disks at each transaction commit, and setting this parameter to a large value does not significantly improve system performance. Setting this parameter to hundreds of megabytes can improve the disk writing performance on the server, to which a large number of transactions are committed. Based on experiences, the default value meets user requirements in most cases.

commit_delay

Parameter description: Specifies the duration of committed data be stored in the WAL buffer.

Type: USERSET

Value range: an integer, ranging from 0 to 100000 (unit: μs). **0** indicates no delay.

Default value: **0**

NOTICE

- When this parameter is set to a value other than 0, the committed transaction is stored in the WAL buffer instead of being written to the WAL immediately. Then, the WalWriter 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 transactions are waiting to be submitted, the delay is a waste of time.

commit_siblings

Parameter description: Specifies a limit on the number of ongoing transactions. If the number of ongoing transactions is greater than the limit, a new transaction will wait for the period of time specified by **commit_delay** before it is submitted. If the number of ongoing transactions is less than the limit, the new transaction is immediately written into a WAL.

Type: USERSET

Value range: an integer ranging from 0 to 1000

Default value: **5**

enable_xlog_group_insert

Parameter description: Specifies whether to enable the group insertion mode for WALs. Only the Kunpeng architecture supports this parameter.

Type: SIGHUP

Value range: Boolean

- **on**: enabled
- **off**: disabled

Default value: **on**

wal_compression

Parameter description: Specifies whether to compress FPI pages.

Type: USERSET

Value range: Boolean

- **on**: enable the compression
- **off**: disable the compression

Default value: **on**

NOTICE

- Only zlib compression algorithm is supported.
- For clusters that are upgraded to the current version from an earlier version, this parameter is set to **off** by default. You can run the **gs_guc** command to enable the FPI compression function if needed.
- If the current version is a newly installed version, this parameter is set to **on** by default.
- If this parameter is manually enabled for a cluster upgraded from an earlier version, the cluster cannot be rolled back.

wal_compression_level

Parameter description: Specifies the compression level of zlib compression algorithm when the **wal_compression** parameter is enabled.

Type: USERSET

Value range: an integer ranging from 0 to 9.

- **0** indicates no compression.
- **1** indicates the lowest compression ratio.
- **9** indicates the highest compression ratio.

Default value: **9**

15.7.2 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.

Type: SIGHUP

Value range: an integer. The minimum value is **1**.

Default value: **64**

NOTICE

Increasing the value of this parameter speeds up the export of big data. Set this parameter based on **checkpoint_timeout** and **shared_buffers**. This parameter affects the number of WAL log 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 log segment files which will be later used.

checkpoint_timeout

Parameter description: Specifies the maximum time between automatic WAL checkpoints.

Type: SIGHUP

Value range: an integer ranging from 30 to 3600 (s)

Default value: **15min**

NOTICE

If the value of **checkpoint_segments** is increased, you need to increase the value of this parameter. The increase of them further requires the increase of **shared_buffers**. Consider all these parameters during setting.

15.8 HA Replication

15.8.1 Sending Server

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 server obtains log files from the primary server for streaming replication.

Type: SIGHUP

Value range: an integer ranging from 2 to INT_MAX

Default value: **128**

Setting suggestions:

- During WAL archiving or recovery from a checkpoint on the server, the system retains more log files than the number specified by **wal_keep_segments**.
- If this parameter is set to a too small value, a transaction log may have been overwritten by a new transaction log before requested by the standby server. As a result, the request fails, and the relationship between the primary and standby servers is interrupted.
- 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.

max_build_io_limit

Parameter description: Specifies the data volume that can be read from the disk per second when the primary server provides a build session to the standby server.

Type: SIGHUP

Value range: an integer ranging from 0 to 1048576. The unit is KB.

Default value: 0, indicating that the I/O flow is not restricted when the primary server provides a build session to the standby server.

Setting suggestions: Set this parameter based on the disk bandwidth and job model. If there is no flow restriction or job interference, for disks with good performance such as SSDs, a full build consumes a relatively small proportion of bandwidth and has little impact on service performance. In this case, you do not need to set the threshold. If the service performance of a common 10,000 rpm SAS disk deteriorates significantly during a build, you are advised to set the parameter to 20 MB.

This setting directly affects the build speed and completion time. Therefore, you are advised to set this parameter to a value larger than 10 MB. During off-peak hours, you are advised to remove the flow restriction to restore to the normal build speed.

NOTE

- This parameter is used during peak hours or when the disk I/O pressure of the primary server is high. It limits the build flow rate on the standby server to reduce the impact on primary server services. After the service peak hours, you can remove the restriction or reset the flow rate threshold.
- You are advised to set a proper threshold based on service scenarios and disk performance.

15.8.2 Primary Server

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.

Type: SIGHUP

Value range: an integer ranging from 0 to 1000000. **0** means no delay.

Default value: **0**

data_replicate_buffer_size

Parameter description: Specifies the size of memory used by queues when the sender sends data pages to the receiver. The value of this parameter affects the buffer size copied for the replication between the primary and standby servers.

Type: POSTMASTER

Value range: an integer ranging from 4 to 1023. The unit is MB.

Default value: **16MB** for CNs and **128MB** for DNs

enable_data_replicate

Parameter description: Specifies the data synchronization mode between the primary and standby servers when data is imported to row-store tables in a database.

Type: USERSET

Value range: Boolean

- **on** indicates that data pages are used for the data synchronization between the primary and standby servers when data is imported to row-store tables in a database. This parameter cannot be set to **on** if **replication_type** is set to **1**.
- **off** indicates that the primary and standby servers synchronize data using Xlogs while the data is imported to a row-store table.

Default value: **on**

enable_incremental_catchup

Parameter description: Specifies the data catchup mode between the primary and standby nodes.

Type: SIGHUP

Value range: Boolean

- **on** indicates that the standby node uses the incremental catchup mode. That is, the standby server scans local data files on the standby server 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 performs catchup between the primary and standby nodes.

Default value: **on**

wait_dummy_time

Parameter description: Specifies the maximum duration for the primary, standby, and secondary clusters to wait for the secondary cluster to start in sequence and the maximum duration for the secondary cluster to send the scanning list when incremental data catchup is enabled.

Type: SIGHUP

Value range: Integer, from 1 to **INT_MAX**, in seconds.

Default value: **300s**

CAUTION

The unit can only be second.

15.9 Query Planning

15.9.1 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** configuration parameter, and adding the statistics collected in a specific column using **ALTER TABLE SET STATISTICS**.

enable_bitmapscan

Parameter description: Controls whether the query optimizer uses the bitmap-scan plan type.

Type: USERSET

Value range: Boolean

- **on** indicates it is enabled.
- **off** indicates it is disabled.

Default value: **on**

enable_hashagg

Parameter description: Controls whether the query optimizer uses the Hash aggregation plan type.

Type: USERSET

Value range: Boolean

- **on** indicates it is enabled.
- **off** indicates it is disabled.

Default value: **on**

enable_hashjoin

Parameter description: Controls whether the query optimizer uses the Hash-join plan type.

Type: USERSET

Value range: Boolean

- **on** indicates it is enabled.
- **off** indicates it is disabled.

Default value: **on**

enable_indexscan

Parameter description: Controls whether the query optimizer uses the index-scan plan type.

Type: USERSET

Value range: Boolean

- **on** indicates it is enabled.
- **off** indicates it is disabled.

Default value: **on**

enable_indexonlyscan

Parameter description: Controls whether the query optimizer uses the index-only-scan plan type.

Type: USERSET

Value range: Boolean

- **on** indicates it is enabled.
- **off** indicates it is disabled.

Default value: **on**

enable_material

Parameter description: Controls whether the query optimizer uses materialization. It is impossible to suppress materialization entirely, but setting this parameter to **off** prevents the optimizer from inserting materialized nodes.

Type: USERSET

Value range: Boolean

- **on** indicates it is enabled.
- **off** indicates it is disabled.

Default value: **on**

enable_mergejoin

Parameter description: Controls whether the query optimizer uses the merge-join plan type.

Type: USERSET

Value range: Boolean

- **on** indicates it is enabled.
- **off** indicates it is disabled.

Default value: **off**

enable_nestloop

Parameter description: Controls 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 parameter to **off** allows the optimizer to choose other methods if available.

Type: USERSET

Value range: Boolean

- **on** indicates it is enabled.
- **off** indicates it is disabled.

Default value: **off**

enable_index_nestloop

Parameter description: Controls whether the query optimizer uses the nested-loop join plan type to scan the parameterized indexes of internal tables.

Type: USERSET

Value range: Boolean

- **on** indicates the query optimizer uses the nested-loop join plan type.
- **off** indicates the query optimizer does not use the nested-loop join plan type.

Default value: The default value for a newly installed cluster is **on**. If the cluster is upgraded from R8C10, the forward compatibility is retained. If the version is upgraded from R7C10 or an earlier version, the default value is **off**.

enable_seqscan

Parameter description: Controls whether the query optimizer uses the sequential scan plan type. It is impossible to suppress sequential scans entirely, but setting this variable to **off** allows the optimizer to preferentially choose other methods if available.

Type: USERSET

Value range: Boolean

- **on** indicates it is enabled.
- **off** indicates it is disabled.

Default value: **on**

enable_sort

Parameter description: Controls whether the query optimizer uses the sort method. It is impossible to suppress explicit sorts entirely, but setting this variable to **off** allows the optimizer to preferentially choose other methods if available.

Type: USERSET

Value range: Boolean

- **on** indicates it is enabled.
- **off** indicates it is disabled.

Default value: **on**

enable_tidscan

Parameter description: Controls whether the query optimizer uses the Tuple ID (TID) scan plan type.

Type: USERSET

Value range: Boolean

- **on** indicates it is enabled.
- **off** indicates it is 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.

Type: SUSET

Value range: Boolean

- **on** indicates the unlocking is allowed.
- **off** indicates the unlocking is not allowed.

Default value: **off**

enforce_oracle_behavior

Parameter description: Controls the rule matching modes of regular expressions.

Type: USERSET

Value range: Boolean

- **on** indicates that the ORACLE matching rule is used.
- **off** indicates that the POSIX matching rule is used.

Default value: **on**

enable_stream_concurrent_update

Parameter description: Controls the use of **stream** in concurrent updates. This parameter is restricted by the **enable_stream_operator** parameter.

Type: USERSET

Value range: Boolean

- **on** indicates that the optimizer can generate stream plans for the **UPDATE** statement.

- **off** indicates that the optimizer can generate only non-stream plans for the **UPDATE** statement.

Default value: **on**

enable_stream_ctescan

Parameter description: Specifies whether a stream plan supports **ctescan**.

Type: USERSET

Value range: Boolean

- **on** indicates that **ctescan** is supported for the stream plan.
- **off** indicates that **ctescan** is not supported for the stream plan.

Default value: **off**

NOTE

In clusters prior to 8.1.3.333, this parameter is automatically enabled. However, in newly installed 8.1.3.333 clusters, this parameter is disabled by default. In upgrade scenarios, this parameter is forward compatible, meaning that its default value remains the same as the cluster's default value before the upgrade.

enable_stream_operator

Parameter description: Controls whether the query optimizer uses streams.

Type: USERSET

Value range: Boolean

- **on** indicates it is enabled.
- **off** indicates it is disabled.

Default value: **on**

enable_stream_recursive

Parameter description: Specifies whether to push **WITH RECURSIVE** join queries to DNs for processing.

Type: USERSET

Value range: Boolean

- **on**: **WITH RECURSIVE** join queries will be pushed down to DNs.
- **off**: **WITH RECURSIVE** join queries will not be pushed down to DNs.

Default value: **on**

max_recursive_times

Parameter description: Specifies the maximum number of **WITH RECURSIVE** iterations.

Type: USERSET

Value range: an integer ranging from 0 to INT_MAX

Default value: **200**

enable_vector_engine

Parameter description: Controls whether the query optimizer uses the vectorized executor.

Type: USERSET

Value range: Boolean

- **on** indicates it is enabled.
- **off** indicates it is disabled.

Default value: **on**

enable_broadcast

Parameter description: Controls whether the query optimizer uses the broadcast distribution method when it evaluates the cost of stream.

Type: USERSET

Value range: Boolean

- **on** indicates it is enabled.
- **off** indicates it is 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.

Type: USERSET

Value range: Boolean

- **on** indicates it is enabled.
- **off** indicates it is disabled.

Default value: **off**

enable_fstream

Parameter description: Controls whether the query optimizer uses streams when it delivers statements. This parameter is only used for external HDFS tables.

This parameter has been discarded. To reserve forward compatibility, set this parameter to **on**, but the setting does not make a difference.

Type: USERSET

Value range: Boolean

- **on** indicates it is enabled.
- **off** indicates it is disabled.

Default value: **off**

best_agg_plan

Parameter description: The query optimizer generates three plans for the aggregate operation under the stream:

1. hashagg+gather(redistribute)+hashagg
2. redistribute+hashagg(+gather)
3. hashagg+redistribute+hashagg(+gather).

This parameter is used to control the query optimizer to generate which type of hashagg plans.

Type: USERSET

Value range: an integer ranging from 0 to 3.

- When the value is set to **1**, the first plan is forcibly generated.
- When the value is set to **2** and if the **group by** column can be redistributed, the second plan is forcibly generated. Otherwise, the first plan is generated.
- When the value is set to **3** and if the **group by** column can be redistributed, the third plan is generated. Otherwise, the first plan is generated.
- When the value is set to **0**, the query optimizer chooses the most optimal plan based on the estimated costs of the three plans above.

Default value: **0**

agg_redistribute_enhancement

Parameter description: When the aggregate operation is performed, which contains multiple **group by** columns and all of the columns are not in the distribution column, you need to select one **group by** column for redistribution. This parameter controls the policy of selecting a redistribution column.

Type: USERSET

Value range: Boolean

- **on** indicates the column that can be redistributed and evaluates the most distinct value for redistribution.
- **off** indicates the first column that can be redistributed for redistribution.

Default value: **off**

enable_valuepartition_pruning

Parameter description: Specifies whether the DFS partitioned table is dynamically or statically optimized.

Type: USERSET

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**

expected_computing_nodegroup

Parameter description: Specifies a computing Node Group or the way to choose such a group. The Node Group mechanism is now for internal use only. You do not need to set it.

During join or aggregation operations, a Node Group can be selected in four modes. In each mode, the specified candidate computing Node Groups are listed for the optimizer to select an appropriate one for the current operator.

Type: USERSET

Value range: a string

- **optimal**: The list of candidate computing Node Groups consists of the Node Group where the operator's operation objects are located and the DNs in the Node Groups on which the current user has the COMPUTE permission.
- **query**: The list of candidate computing Node Groups consists of the Node Group where the operator's operation objects are located and the DNs in the Node Groups where base tables involved in the query are located.
- **bind**: If the current session user is a logical cluster user, the candidate computing Node Group is the Node Group of the logical cluster associated with the current user. If the session user is not a logical cluster user, the candidate computing Node Group selection rule is the same as that when this parameter is set to **query**.
- Node Group name:
  - If **enable_nodegroup_debug** is set to **off**, the list of candidate computing Node Groups consists of the Node Group where the operator's operation objects are located and the specified Node Group.
  - If **enable_nodegroup_debug** is set to **on**, the specified Node Group is used as the candidate Node Group.

Default value: **bind**

enable_nodegroup_debug

Parameter description: Specifies whether the optimizer assigns computing workloads to a specific Node Group when multiple Node Groups exist in an environment. The Node Group mechanism is now for internal use only. You do not need to set it.

This parameter takes effect only when **expected_computing_nodegroup** is set to a specific Node Group.

Type: USERSET

Value range: Boolean

- **on** indicates that computing workloads are assigned to the Node Group specified by **expected_computing_nodegroup**.
- **off** indicates no Node Group is specified to compute.

Default value: **off**

stream_multiple

Parameter description: Specifies the weight used for optimizer to calculate the final cost of stream operators.

The base stream cost is multiplied by this weight to make the final cost.

Type: USERSET

Value range: a floating point number ranging from 0 to DBL_MAX

Default value: **1**

NOTICE

This parameter is applicable only to Redistribute and Broadcast streams.

qrw_inlist2join_optmode

Parameter description: Specifies whether enable inlist-to-join (inlist2join) query rewriting.

Type: USERSET

Value range: a string

- **disable**: inlist2join disabled
- **cost_base**: cost-based inlist2join query rewriting
- **rule_base**: forcible rule-based inlist2join query rewriting
- A positive integer: 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

Type: USERSET

Value range: a string

- **off**: policy disabled
- **normal**: radical policy. All possible skews are optimized.
- **lazy**: conservative policy. Uncertain skews are ignored.

Default value: normal

15.9.2 Optimizer Cost Constants

This section describes the optimizer cost constants. The cost variables described in this section are measured on an arbitrary scale. Only their relative values matter, therefore scaling them all in or out by the same factor will result in no differences 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.

Type: USERSET

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.

Type: USERSET

Value range: a floating point number ranging from 0 to DBL_MAX

Default value: **4**

NOTE

- Although the server allows you to set the value of **random_page_cost** to 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.
- This value can be overwritten for tables and indexes in a particular tablespace by setting the tablespace parameter of the same name.
- Comparing to **seq_page_cost**, reducing this value will cause the system to prefer index scans and raising it makes index scans relatively more expensive. You can increase or decrease both values at the same time to change the disk I/O cost relative to CPU cost.

cpu_tuple_cost

Parameter description: Specifies the optimizer's estimated cost of processing each row during a query.

Type: USERSET

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.

Type: USERSET

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 during a query.

Type: USERSET

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(DWS)'s shared buffer 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(DWS). 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.

Type: USERSET

Value range: an integer ranging is from 1 to INT_MAX. The unit is 8 KB.

A value greater than the default one may enable index scanning, and a value less than the default one may enable sequence scanning.

Default value: **128 MB**

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.

Type: USERSET

Value range: a floating point number ranging from 0 to DBL_MAX

Default value: **0**

15.9.3 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 cost of producing plans are sometimes inferior to those found by the normal exhaustive-search algorithm.

geqo

Parameter description: Controls the use of genetic query optimization.

Type: USERSET

Value range: Boolean

- **on** indicates GEQO is enabled.
- **off** indicates GEQO is disabled.

Default value: **on**

NOTICE

Generally, do not set this parameter to **off**. **geqo_threshold** provides more subtle control of GEQO.

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.

Type: USERSET

Value range: an integer ranging from 2 to INT_MAX

Default value: **12**

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.

geqo_effort

Parameter description: Controls the trade-off between planning time and query plan quality in GEQO.

Type: USERSET

Value range: an integer ranging from 1 to 10

Default value: **5**

NOTICE

- Larger values increase the time spent in query planning, but also increase the probability that an efficient query plan is chosen.
- **geqo_effort** does not have direct effect. This parameter is only used to compute the default values for the other variables that influence GEQO behavior. You can manually set other parameters as required.

geqo_pool_size

Parameter description: Specifies the pool size used by GEQO, that is, the number of individuals in the genetic population.

Type: USERSET

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(DWS) selects a proper value based on **geqo_effort** and the number of tables.

Default value: **0**

geqo_generations

Parameter description: Specifies the number parameter iterations of the algorithm used by GEQO.

Type: USERSET

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.

Type: USERSET

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.

Type: USERSET

Value range: a floating point number ranging from 0.0 to 1.0

NOTICE

Varying the value changes the setting of join paths explored, and may result in a better or worse path being found.

Default value: 0

15.9.4 Other Optimizer Options

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%. During sampling, the random sampling size is **default_statistics_target** x 300. For example, if the **default_statistics_target** is 100, 30,000 data records from 30,000 pages are randomly sampled.

Type: USERSET

Value range: an integer ranging from -100 to 10000

NOTICE

- A larger positive number than the parameter 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 optimize the query plan.
- 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 autoanalyze function does not support percentage sampling. The sampling uses the default value of this parameter.
- If this parameter is set to a positive value, you must have the **ANALYZE** permission to execute **ANALYZE**.
- If this parameter is set to a negative value, that is, percentage sampling, you need to be granted the ANALYZE and SELECT permissions to execute ANALYZE.

Default value: **100**

random_function_version

Parameter description: Specifies the random function version selected by ANALYZE during data sampling. This feature is supported only in 8.1.2 or later.

Type: USERSET

Value range: enumerated values

- The value **0** indicates that the random function provided by the C standard library is used.
- The value **1** indicates that the optimized and enhanced random function is used.

Default value: **0**

constraint_exclusion

Parameter description: Controls the query optimizer's use of table constraints to optimize queries.

Type: USERSET

Value range: enumerated values

- **on** indicates the constraints for all tables are examined.
- **off**: No constraints are examined.
- **partition** indicates that only constraints for inherited 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, this parameter is set to **on** by default to partition tables. If this parameter is set to **on**, extra planning is imposed on simple queries, which has no benefits. 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.

Type: USERSET

Value range: a floating point number ranging from 0.0 to 1.0

NOTICE

Smaller values than the default value 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.

Type: USERSET

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.

Type: USERSET

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. **OPTIMIZE_PLAN** indicates the optimal plan, the cost of which is estimated using the dynamic planning algorithm, and its value is **0**. **RANDOM_PLAN** indicates the plan that is randomly generated. If **plan_mode_seed** is set to **-1**, you do not need to specify the value of the seed identifier. Instead, the optimizer generates a random integer ranging from **1** to **2147483647**, and then generates a random execution plan based on this random number. If **plan_mode_seed** is set to an integer ranging from **1** to **2147483647**, you need to specify the value of the seed identifier, and the optimizer generates a random execution plan based on the seed value.

Type: USERSET

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 different random execution plans, which may not be the optimal. 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 hint will not be used.

enable_hdfs_predicate_pushdown

Parameter description: Specifies whether the function of pushing down predicates the native data layer is enabled.

Type: SUSET

Value range: Boolean

- **on** indicates this function is enabled.
- **off** indicates this function is disabled.

Default value: **on**

enable_random_datanode

Parameter description: Specifies whether the function that random query about DNs in the replication table is enabled. A complete data table is stored on each DN for random retrieval to release the pressure on nodes.

Type: USERSET

Value range: Boolean

- **on**: This function is enabled.
- **off**: This function is disabled.

Default value: **on**

hashagg_table_size

Parameter description: Specifies the hash table size during **HASH AGG** execution.

Type: USERSET

Value range: an integer ranging from 0 to INT_MAX/2

Default value: **0**

enable_codegen

Parameter description: Specifies whether code optimization can be enabled. Currently, the code optimization uses the LLVM optimization.

Type: USERSET

Value range: Boolean

- **on** indicates code optimization can be enabled.
- **off** indicates code optimization cannot be enabled.

NOTICE

Currently, the LLVM optimization only supports the vectorized executor and SQL on Hadoop features. You are advised to set this parameter to **off** in other cases.

Default value: **on**

codegen_strategy

Parameter description: Specifies the codegen optimization strategy that is used when an expression is converted to codegen-based.

Type: USERSET

Value range: enumerated values

- **partial** indicates that you can still call the LLVM dynamic optimization strategy using the codegen framework of an expression even if functions that are not codegen-based exist in the expression.
- **pure** indicates that the LLVM dynamic optimization strategy can be called only when all functions in an expression can be codegen-based.

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.

Type: USERSET

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.

Type: USERSET

Value range: an integer ranging from **0** to **INT_MAX**

Default value: **10000**

enable_constraint_optimization

Parameter description: Specifies whether the informational constraint optimization execution plan can be used for an HDFS foreign table.

Type: SUSET

Value range: Boolean

- **on** indicates the plan can be used.
- **off** indicates the plan cannot be used.

Default value: **on**

enable_bloom_filter

Parameter description: Specifies whether the BloomFilter optimization is used.

Type: USERSET

Value range: Boolean

- **on** indicates the BloomFilter optimization can be used.
- **off** indicates the BloomFilter optimization cannot be used.

Default value: **on**

NOTICE

Scenario: If in a HASH JOIN, the thread of the foreign table contains HDFS tables or column-store tables, the Bloom filter is triggered.

Constraints:

1. Only **INNER JOIN**, **SEMI JOIN**, **RIGHT JOIN**, **RIGHT SEMI JOIN**, **RIGHT ANTI JOIN** and **RIGHT ANTI FULL JOIN** are supported.
2. The number of rows in the internal table in the join cannot exceed 50,000.
3. JOIN condition of the internal table: It cannot be an expression for HDFS internal or foreign tables. It can be an expression for column-store tables, but only at the non-join layer.
4. The join condition of the foreign table must be simple column join.
5. When the join conditions of the internal and foreign tables (HDFS) are both simple column joins, the estimated data that can be removed at the plan layer must be over 1/3.
6. Joined columns cannot contain NULL values.
7. Data is not flushed to disks at the JOIN layer.
8. Data type:
   - HDFS internal and foreign tables support SMALLINT, INTEGER, BIGINT, REAL/FLOAT4, DOUBLE PRECISION/FLOAT8, CHAR(n)/CHARACTER(n)/NCHAR(n), VARCHAR(n)/CHARACTER VARYING(n), CLOB and TEXT.
   - Column-store tables support SMALLINT, INTEGER, BIGINT, OID, "char", CHAR(n)/CHARACTER(n)/NCHAR(n), VARCHAR(n)/CHARACTER VARYING(n), NVARCHAR2(n), CLOB, TEXT, DATE, TIME, TIMESTAMP and TIMESTAMPTZ. The collation of the character type must be **C**.

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.

Type: USERSET

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 allow automatic statistics collection for a table that has no statistics or a table whose amount of data modification reaches the threshold for triggering **ANALYZE** when a plan is generated. In this case, **AUTOANALYZE** cannot be triggered for foreign tables or temporary tables with the **ON COMMIT [DELETE ROWS|DROP]** option. To collect statistics, you need to 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.

NOTICE

If the amount of data modification reaches the threshold for triggering **ANALYZE**, the amount of data modification exceeds **autovacuum_analyze_threshold** + **autovacuum_analyze_scale_factor** * *reltuples*. *reltuples* indicates the estimated number of rows in the table recorded in **pg_class**.

Type: SUSET

Value range: Boolean

- **on** indicates that the table statistics are automatically collected.
- **off** indicates that the table statistics are not automatically collected.

Default value: **on**

query_dop

Parameter description: Specifies the user-defined degree of parallelism.

Type: USERSET

Value range: an integer ranging from -64 to 64.

[1, 64]: Fixed SMP is enabled, and the system will use the specified degree.

0: SMP adaptation is enabled. The system dynamically selects the optimal degree of parallelism for each query, ranging from 1 to 8 on x86 platforms and 1 to 64 on Kunpeng platforms, based on resource usage and query plans.

[-64, -1]: SMP adaptation is enabled, and the system will dynamically select a degree from the limited range.

NOTE

- For TP services that mainly involve short queries, if services cannot be optimized through lightweight CNs or statement delivery, it will take a long time to generate an SMP plan. You are advised to set **query_dop** to **1**. For AP services with complex statements, you are advised to set **query_dop** to **0**.
- After enabling concurrent queries, ensure you have sufficient CPU, memory, network, and I/O resources to achieve the optimal performance.
- To prevent performance deterioration caused by an overly large value of **query_dop**, the system calculates the maximum number of available CPU cores for a DN and uses the number as the upper limit for this parameter. If the value of **query_dop** is greater than 4 and also the upper limit, the system resets **query_dop** to the upper limit.

Default value: **1**

query_dop_ratio

Parameter description: Specifies the DOP multiple used to adjust the optimal DOP preset in the system when **query_dop** is set to **0**. That is, DOP = Preset DOP x query_dop_ratio (ranging from 1 to 64). If this parameter is set to **1**, the DOP cannot be adjusted.

Type: USERSET

Value range: a floating point number ranging from 0 to 64

Default value: **1**

debug_group_dop

Parameter description: Specifies the unified DOP parallelism degree allocated to the groups that use the Stream operator as the vertex in the generated execution plan when the value of **query_dop** is **0**. This parameter is used to manually specify the DOP for specific groups for performance optimization. Its format is **G1,D1,G2,D2,...,**, where **G1** and **G2** indicate the group IDs that can be obtained from logs and **D1** and **D2** indicate the specified DOP values and can be any positive integers.

Type: USERSET

Value range: a string

Default value: empty

NOTICE

This parameter is used only for internal optimization and cannot be set. You are advised to use the default value.

enable_analyze_check

Parameter description: Checks whether statistics were collected about tables whose **reltuples** and **relpages** are shown as **0** in **pg_class** during plan generation. **This parameter is no longer used in cluster versions 8.1.3 and later, but is reserved for compatibility with earlier versions. The setting of this parameter does not take effect.**

Type: SUSET

Value range: Boolean

- **on** enables the check.
- **off** disables the check.

Default value: **on**

enable_sonic_hashagg

Parameter description: Specifies whether to use the Hash Agg operator for column-oriented hash table design when certain constraints are met.

Type: USERSET

Value range: Boolean

- **on** indicates that the Hash Agg operator is used for column-oriented hash table design when certain constraints are met.
- **off** indicates that the Hash Agg operator is not used for column-oriented hash table design.

NOTE

- If **enable_sonic_hashagg** is enabled and certain constraints are met, the Hash Agg operator will be used for column-oriented hash table design, and the memory usage of the operator can be reduced. However, in scenarios where the code generation technology (enabled by **enable_codegen**) can significantly improve performance, 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**.

Default value: **on**

enable_sonic_hashjoin

Parameter description: Specifies whether to use the Hash Join operator for column-oriented hash table design when certain constraints are met.

Type: USERSET

Value range: Boolean

- **on** indicates that the Hash Join operator is used for column-oriented hash table design when certain constraints are met.
- **off** indicates that the Hash Join operator is not used for column-oriented hash table design.

NOTE

- Currently, the parameter can be used only for Inner Join.
- If **enable_sonic_hashjoin** is enabled, the memory usage of 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**.

Default value: **on**

enable_sonic_optspill

Parameter description: Specifies whether to optimize the number of hash join or hash agg files flushed to disks in the sonic scenario. This parameter takes effect only when **enable_sonic_hashjoin** or **enable_sonic_hashagg** is enabled.

Type: USERSET

Value range: Boolean

- **on** indicates that the number of files flushed to disks is optimized.
- **off** indicates that the number of files flushed to disks is not optimized.

NOTE

For the hash join or hash agg operator that meets the sonic criteria, if this parameter is set to **off**, one file is flushed to disks for each column. If this parameter is set to **on** and the data types of different columns are similar, only one file (a maximum of five files) will be flushed to disks.

Default value: **on**

expand_hashtable_ratio

Parameter description: Specifies the expansion ratio used to resize the hash table during the execution of the Hash Agg and Hash Join operators.

Type: USERSET

Value range: a floating point number of 0 or ranging from 0.5 to 10

NOTE

- Value **0** indicates that the hash table is adaptively expanded based on the current memory size.
- The value ranging from 0.5 to 10 indicates the multiple used to expand the hash table. Generally, a larger hash table delivers better performance but occupies more memory space. If the memory space is insufficient, data may be spilled to disks in advance, causing performance deterioration.

Default value: **0**

plan_cache_mode

Parameter description: Specifies the policy for generating an execution plan in the **prepare** statement.

Type: USERSET

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 a **prepare** statement where parameters in the execute statement is embedded in the **prepare** statement. 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 for 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 solution 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 parameter field. When some bound parameters are used, the plan execution performance is poor.

Default value: **auto**

wlm_query_accelerate

Parameter description: Specifies whether the query needs to be accelerated when short query acceleration is enabled.

Type: USERSET

Value range: an integer ranging from **–1** to **1**

- **-1**: indicates that short queries are controlled by the fast lane, and the long queries are controlled by the slow lane.
- **0**: indicates that queries are not accelerated. Both short and long queries are controlled by the slow lane.
- **1**: indicates that queries are accelerated. Both short queries and long queries are controlled by the fast lane.

Default value: **–1**

show_unshippable_warning

Parameter description: Specifies whether to print the alarm for the statement pushdown failure to the client.

Type: USERSET

Value range: Boolean

- **on**: Records the reason why the statement cannot be pushed down in a WARNING log and prints the log to the client.
- **off**: Logs the reason why the statement cannot be pushed down only.

Default value: **off**

hashjoin_spill_strategy

Parameter description: specifies the hash join policy for flushing data to disks. This feature is supported in 8.1.2 or later.

Type: USERSET

Value range: The value is an integer ranging from 0 to 4.

- **0**: If the size of the inner table is large and cannot be partitioned after data is flushed to disks for multiple times, the system attempts to place the outer table in the available memory of the database to create a hash table. If both the inner and outer tables are large, a nested loop join is performed.
- **1**: If the size of the inner table is large and cannot be partitioned after data is flushed to disks for multiple times, the system attempts to place the outer table in the available memory of the database to create a hash table. If both the inner and outer tables are large, a hash join is forcibly performed.
- **2**: If the size of the inner table is large and cannot be partitioned after data is flushed to disks for multiple times, a hash join is forcibly performed.
- **3**: If the size of the inner table is large and cannot be partitioned after data is flushed to disks for multiple times, the system attempts to place the outer table in the available memory of the database to create a hash table. If both the inner and outer tables are large, an error is reported.
- **4**: If the size of the inner table is large and cannot be partitioned after data is flushed to disks for multiple times, an error is reported.

NOTE

- This parameter is valid only for a vectorized hash join operator.
- If the number of distinct values is small and the data volume is large, data may fail to be flushed to disks. As a result, the memory usage is too high and the memory is out of control. If this parameter is set to **0**, the system attempts to swap the inner and outer tables or perform a nested loop join to prevent this problem. However, a nested loop join may deteriorate performance in some scenarios.
- The value **0** does not take effect for a vectorized full join, and the behavior is the same as that of the value **1**. The system attempts to create a hash table only for the outer table and does not perform a nested loop join.

Default value: **0**

max_streams_per_query

Parameter description: Controls the number of Stream nodes in a query plan. (This parameter is supported only in 8.1.3.200 and later cluster versions.)

Type: SUSET

Value range: an integer ranging from –1 to 10000.

- **-1** indicates that the number of Stream nodes in the query plan is not limited.
- A value within the range **0** to **10000** indicates that when the number of Stream nodes in the query plan exceeds the specified value, an error is reported and the query plan will not be executed.

NOTE

- This parameter controls only the Stream nodes on DNs and does not control the Gather nodes on the CN.
- This parameter does not affect the EXPLAIN query plan, but affects EXPLAIN ANALYZE and EXPLAIN PERFORMANCE.

Default value: **–1**

15.10 Error Reporting and Logging

15.10.1 Logging Time

client_min_messages

Parameter description: Specifies which level of messages are sent to the client. Each level covers all the levels following it. The lower the level is, the fewer messages are sent.

Type: USERSET

NOTICE

When the values of **client_min_messages** and **log_min_messages** are the same, the levels are different.

Valid values: Enumerated values. Valid values: **debug5**, **debug4**, **debug3**, **debug2**, **debug1**, **info**, **log**, **notice**, **warning**, **error** For details about the parameters, see **Table 15-3**.

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.

Type: SUSET

NOTICE

When the values of **client_min_messages** and **log_min_messages** are the same, the levels are different.

Value range: enumerated type. Valid values: **debug5**, **debug4**, **debug3**, **debug2**, **debug1**, **info**, **log**, **notice**, **warning**, **error**, **fatal**, **panic** For details about the parameters, see **Table 15-3**.

Default value: **warning**

log_min_error_statement

Parameter description: Specifies which SQL statements that cause errors condition will be recorded in the server log.

Type: SUSET

Value range: enumerated type. Valid values: **debug5**, **debug4**, **debug3**, **debug2**, **debug1**, **info**, **log**, **notice**, **warning**, **error**, **fatal**, **panic** For details about the parameters, see **Table 15-3**.

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 statement execution durations. The execution duration that is greater than the specified value will be logged.

This parameter helps track query statements that need to be optimized. For clients using extended query protocol, durations of the Parse, Bind, and Execute are logged independently.

Type: SUSET

NOTICE

If this parameter and **log_statement** are used at the same time, statements recorded based on the value of **log_statement** will not be logged again after their execution duration exceeds the value of this parameter. 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 current statement message to the last logged duration.

Value range: an integer ranging from -1 to INT_MAX. The unit is millisecond.

- If this parameter is set to **250**, execution durations of SQL statements that run 250 ms or longer will be logged.
- **0**: Execution durations of all the statements are logged.
- **–1**: This feature is disabled.

Default value: **30min**

backtrace_min_messages

Parameter description: Prints the function's stack information to the server's log file if the level of information generated is greater than or equal to this parameter level.

Type: SUSET

NOTICE

This parameter is used for locating customer on-site problems. Because frequent stack printing will affect the system's overhead and stability, therefore, when you locate the onsite problems, set the value of this parameter to ranks other than **fatal** and **panic**.

Value range: enumerated values

Valid values: **debug5**, **debug4**, **debug3**, **debug2**, **debug1**, **info**, **log**, **notice**, **warning**, **error**, **fatal**, **panic** For details about the parameters, see **Table 15-3**.

Default value: **panic**

Table 15-3 explains the message security levels used in GaussDB(DWS). If logging output is sent to **syslog** or **eventlog**, severity is translated in GaussDB(DWS) as shown in the table.

Table 15-3 Message Severity Levels

Severity	Description	syslog	eventlog
debug[1-5]	Provides detailed debug information.	DEBUG	INFORMATIO N
log	Reports information of interest to administrators, for example, checkpoint activity.	INFO	INFORMATIO N
info	Provides information implicitly requested by the user, for example, output from **VACUUM VERBOSE**.	INFO	INFORMATIO N
notice	Provides information that might be helpful to users, for example, notice of 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	Reports an error that causes a command to terminate.	WARNING	ERROR
fatal	Reports the reason that causes a session to terminate.	ERR	ERROR

Severity	Description	syslog	eventlog
panic	Reports an error that caused all database sessions to terminate.	CRIT	ERROR

plog_merge_age

Parameter description: Specifies the output interval of performance log data.

Type: SUSET

NOTICE

This parameter value is in milliseconds. You are advised to set this parameter to a value that is a multiple of 1000. That is, the value is in seconds. Name extension of the performance log files controlled by this parameter is .prf. These log files are stored in the **$GAUSSLOG/gs_profile/**<*node_name*> directory. *node_name* is the value of **pgxc_node_name** in the **postgres.conf** file. You are advised not to use this parameter externally.

Value range: an integer ranging from 0 to INT_MAX. The unit is millisecond (ms).

- **0** indicates that the current session will not output performance log data.
- A value other than 0 indicates the output interval of performance log data. As the value decreases, more log data is generated, which negatively impacts performance.

Default value: **3s**

profile_logging_module

Parameter description: Specifies the type of performance logs. When using this parameter, ensure that the value of **plog_merge_age** is not 0. This parameter is a session-level parameter, and you are not advised to use the **gs_guc** tool to set it. Only clusters of 8.1.3 and later versions support this function.

Type: USERSET

Value range: a string

Default value: OBS, HADOOP and REMOTE_DATANODE are enabled. MD is disabled. You can run the **SHOW profile_logging_module** command to view the value.

Setting method: First, you can run **SHOW profile_logging_module** to view which module is controllable. For example, the query output result is as follows:

```
show profile_logging_module;
profile_logging_module
-------------------------------------------
ALL,on(OBS,HADOOP,REMOTE_DATANODE),off(MD)(1 row)
```

Open the MD performance log and view the setting. The ALL identifier is equivalent to a shortcut operation. That is, logs of all modules can be enabled or disabled.

```
set profile_logging_module='on(md)';
SET

show profile_logging_module;
profile_logging_module
---------------------------------------------
ALL,on(MD,OBS,HADOOP,REMOTE_DATANODE),off()(1 row)
```

15.10.2 Logging Content

debug_print_parse

Parameter description: Specifies whether to print parsing tree results.

Type: SIGHUP

Value range: Boolean

- **on** indicates the printing result function is enabled.
- **off** indicates the printing result function is disabled.

Default value: **off**

debug_print_rewritten

Parameter description: Specifies whether to print query rewriting results.

Type: SIGHUP

Value range: Boolean

- **on** indicates the printing result function is enabled.
- **off** indicates the printing result function is disabled.

Default value: **off**

debug_print_plan

Parameter description: Specifies whether to print query execution results.

Type: SIGHUP

Value range: Boolean

- **on** indicates the printing result function is enabled.
- **off** indicates the printing result function 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**, and then provide the information to external maintenance engineers for fault locating. After you finish using the logs, delete them as soon as possible.

debug_pretty_print

Parameter description: Specifies the logs produced by **debug_print_parse**, **debug_print_rewritten**, and **debug_print_plan**. The output format is more readable but much longer than the output generated when this parameter is set to **off**.

Type: USERSET

Value range: Boolean

- **on** indicates the indentation is enabled.
- **off** indicates the indentation is disabled.

Default value: **on**

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.

Type: SUSET

Value range: Boolean

- If this parameter is set to **off**, the difference between setting this parameter and setting **log_min_duration_statement** is that exceeding **log_min_duration_statement** forces the text of the query to be logged, but this parameter does not.
- If this parameter is set to **on** and **log_min_duration_statement** has a positive value, all durations are 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: **on**

log_error_verbosity

Parameter description: Specifies the amount of detail written in the server log for each message that is logged.

Type: SUSET

Value range: enumerated values

- **terse** indicates that the output excludes the logging of 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 logging of 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_lock_waits

Parameter description: If the time that a session used to wait 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.

Type: SUSET

Value range: Boolean

- **on** indicates the information is recorded.
- **off** indicates the information is not recorded.

Default value: **off**

log_statement

Parameter description: Specifies whether to record SQL statements. 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).

Type: SUSET

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 the extended query protocol is used, this setting also does not log statements before the execution phase (during parse analysis or planning). 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 are recorded. The PREPARE, EXECUTE, and EXPLAIN ANALYZE statements are also recorded.

Default value: **none**

log_temp_files

Parameter description: Specifies whether to record the delete information of temporary files. Temporary files can be created for sorting, hashing, and temporary querying results. A log entry is generated for each temporary file when it is deleted.

Type: SUSET

Value range: an integer ranging from -1 to INT_MAX. The unit is KB.

- A positive value indicates that the delete information of temporary files whose values are larger than that of **log_temp_files** is recorded.
- If the parameter is set to **0**, all the delete information of temporary files is recorded.
- If the parameter is set to **-1**, the delete information of no temporary files is recorded.

Default value: **–1**

logging_module

Parameter description: Specifies whether module logs can be output on the server. This parameter is a session-level parameter, and you are not advised to use the **gs_guc** tool to set it.

Type: USERSET

Value range: a string

Default value: **off**. All the module logs on the server can be viewed by running **show logging_module**.

Setting method: First, you can run **show logging_module** to view which module is controllable. For example, the query output result is as follows:

```
show logging_module;
logging_module
-----------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------
------------------------------
ALL,on(),off(DFS,GUC,HDFS,ORC,SLRU,MEM_CTL,AUTOVAC,ANALYZE,CACHE,ADIO,SSL,GDS,TBLSPC,WLM,SP
ACE,OBS,EXECUTOR,VEC_EXECUTOR,STREAM,LLVM,OPT,OPT_REWRITE,OPT_JOIN,OPT_AGG,OPT_SUBPLAN,
OPT_SETOP,OPT_CARD,OPT_SKEW,SMP,UDF,COOP_ANALYZE,WLMCP,ACCELERATE,PLANHINT,PARQUET,CARB
ONDATA,SNAPSHOT,XACT,HANDLE,CLOG,TQUAL,EC,REMOTE,CN_RETRY,PLSQL,TEXTSEARCH,SEQ,INSTR,CO
MM_IPC,COMM_PARAM,CSTORE,JOB,STREAMPOOL,STREAM_CTESCAN)
(1 row)
```

Controllable modules are identified by uppercase letters, and the special ID ALL is used for setting all module logs. You can control module logs to be exported by setting the log modules to **on** or **off**. Enable log output for SSL:

```
set logging_module='on(SSL)';
SET
show
logging_module;

                                                                   logging_module
--------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------

ALL,on(SSL),off(DFS,GUC,HDFS,ORC,SLRU,MEM_CTL,AUTOVAC,ANALYZE,CACHE,ADIO,GDS,TBLSPC,WLM,SP
ACE,OBS,EXECUTOR,VEC_EXECUTOR,STREAM,LLVM,OPT,OPT_REWRITE,OPT_JOIN,OPT_AGG,OPT_SUBPLAN,
OPT_SETOP,OPT_CARD,OPT_SKEW,SMP,UDF,COOP_ANALYZE,WLMCP,A
CCELERATE,PLANHINT,PARQUET,CARBONDATA,SNAPSHOT,XACT,HANDLE,CLOG,TQUAL,EC,REMOTE,CN_RET
RY,PLSQL,TEXTSEARCH,SEQ,INSTR,COMM_IPC,COMM_PARAM,CSTORE,JOB,STREAMPOOL,STREAM_CTESCA
N)
(1 row)
```

SSL log output is enabled.

The ALL identifier is equivalent to a shortcut operation. That is, logs of all modules can be enabled or disabled.

```
set logging_module='off(ALL)';
SET
show
logging_module;

                                      logging_module
--------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------

ALL,on(),off(DFS,GUC,HDFS,ORC,SLRU,MEM_CTL,AUTOVAC,ANALYZE,CACHE,ADIO,SSL,GDS,TBLSPC,WLM,SP
ACE,OBS,EXECUTOR,VEC_EXECUTOR,STREAM,LLVM,OPT,OPT_REWRITE,OPT_JOIN,OPT_AGG,OPT_SUBPLAN,
OPT_SETOP,OPT_CARD,OPT_SKEW,SMP,UDF,COOP_ANALYZE,WLMCP,
ACCELERATE,PLANHINT,PARQUET,CARBONDATA,SNAPSHOT,XACT,HANDLE,CLOG,TQUAL,EC,REMOTE,CN_RE
TRY,PLSQL,TEXTSEARCH,SEQ,INSTR,COMM_IPC,COMM_PARAM,CSTORE,JOB,STREAMPOOL,STREAM_CTESCA
N)
(1 row)

set logging_module='on(ALL)';
SET
show
logging_module;

                       logging_module
--------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------

ALL,on(DFS,GUC,HDFS,ORC,SLRU,MEM_CTL,AUTOVAC,ANALYZE,CACHE,ADIO,SSL,GDS,TBLSPC,WLM,SPACE,
OBS,EXECUTOR,VEC_EXECUTOR,STREAM,LLVM,OPT,OPT_REWRITE,OPT_JOIN,OPT_AGG,OPT_SUBPLAN,OPT_
SETOP,OPT_CARD,OPT_SKEW,SMP,UDF,COOP_ANALYZE,WLMCP,ACCELE
RATE,PLANHINT,PARQUET,CARBONDATA,SNAPSHOT,XACT,HANDLE,CLOG,TQUAL,EC,REMOTE,CN_RETRY,PLS
QL,TEXTSEARCH,SEQ,INSTR,COMM_IPC,COMM_PARAM,CSTORE,JOB,STREAMPOOL,STREAM_CTESCAN),off()
(1 row)
```

COMM_IPC logs must be enabled or disabled explicitly. You can run either of the following command to enable the log function of COMM_IPC:

```
set logging_module='on(ALL)';
SET
set logging_module='on(COMM_IPC)';
SET
```

After the setting is performed, the log function of the COMM_IPC module will not be automatically disabled. To disable the log function of the COMM_IPC module, you must run the following commands:

```
set logging_module='off(ALL)';
SET
set logging_module='off(COMM_IPC)';
SET
```

Dependency relationship: The value of this parameter depends on the settings of **log_min_messages**.

enable_unshipping_log

Parameter description: Specifies whether to log statements that are not pushed down. The logs help locate performance issues that may be caused by statements not pushed down.

Type: SUSET

Value range: Boolean

- **on**: Statements not pushed down will be logged.
- **off**: Statements not pushed down will not be logged.

Default value: **on**

15.11 Alarm Detection

During cluster running, error scenarios can be detected in a timely manner to inform users as soon as possible.

enable_alarm

Parameter description: Enables the alarm detection thread to detect the fault scenarios that may occur in the database.

Type: POSTMASTER

Value range: Boolean

- **on** indicates the alarm detection thread can be enabled.
- **off** indicates the alarm detection thread cannot be enabled.

Default value: **on**

connection_alarm_rate

Parameter description: Specifies the ratio restriction that 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**.

Type: SIGHUP

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.

Type: SIGHUP

Value range: a non-negative integer. The unit is second.

Default value: **10**

15.12 Statistics During the Database Running

15.12.1 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 an 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.

Type: SUSET

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 data activities.

Type: SUSET

Value range: Boolean

- **on** indicates that the statistics collection function is enabled.
- **off** indicates that the statistics collection function is disabled.

NOTE

When the database to be cleaned up is selected from the AutoVacuum automatic cleanup process, the database statistics are required. In this case, the default value is set to **on**.

Default value: **on**

track_io_timing

Parameter description: Collects statistics about I/O invoking timing in the database. The I/O timing statistics can be queried by using the **pg_stat_database** parameter.

Type: SUSET

Value range: Boolean

- If this parameter is set to **on**, the collection function is enabled. In this case, the collector repeatedly queries the OS at the current time. As a result, large numbers 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 about invoking times and duration in a function.

Type: SUSET

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 and C language functions are 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.

Type: POSTMASTER

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 on Windows task manager by running the **ps** command.

Type: SUSET

Value range: Boolean

- **on** indicates that the statistics collection function is enabled.
- **off** indicates that the statistics collection function is disabled.

Default value: **off**

track_thread_wait_status_interval

Parameter description: Specifies the interval of collecting the thread status information periodically.

Type: SUSET

Value range: an integer ranging from 0 to 1440. The unit is minute (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.

Type: USERSET

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

instr_unique_sql_count

Parameter description: Specifies whether to collect Unique SQL statements and the maximum number of collected Unique SQL statements.

Type: SIGHUP

Value range: an integer ranging from 0 to INT_MAX

- If it is set to **0**, Unique SQL statistics are not collected.
- If the value is greater than **0**, the number of Unique SQL statements collected on the CN cannot exceed the value of this parameter. When the number of collected Unique SQL statements reaches the upper limit, the collection is

stopped. In this case, you can increase the value of **reload** to continue the collection.

Default value: **0**

CAUTION

If a new value is less than the original value, the Unique SQL statistics collected on the corresponding CN will be cleared. Note that the clearing operation is performed by the background thread of the resource management module. If the GUC parameter **use_workload_manager** is set to **off**, the clearing operation may fail. In this case, you can use the **reset_instr_unique_sql** function for clearing.

instr_unique_sql_timeout

Parameter description: Specifies the lifetime of a Unique SQL statement. The background thread of StatCollector checks all Unique SQL statements every hour. If a Unique SQL statement is not executed for more than **instr_unique_sql_timeout** hours, the Unique SQL statement will be deleted. This feature is supported in 8.1.2 or later.

Type: SIGHUP

Value range: an integer ranging from **0** to **INT_MAX**. The unit is hour.

- The value **0** indicates that expired Unique SQL statements will not be deleted.
- If the value is greater than **0**, the Unique SQL statement that is not executed for more than **instr_unique_sql_timeout** hours will be deleted.

Default value: **24**

track_sql_count

Parameter description: Specifies whether to collect statistics on the number of the **SELECT**, **INSERT**, **UPDATE**, **DELETE**, and **MERGE INTO** statements that are being executed in each session, the response time of the **SELECT**, **INSERT**, **UPDATE**, and **DELETE** statements, and the number of DDL, DML, and DCL statements.

Type: SUSET

Value range: Boolean

- **on** indicates that the statistics collection function is enabled.
- **off** indicates that the statistics collection 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 when the view **gs_sql_count**, **pgxc_sql_count**, **gs_workload_sql_count**, **pgxc_workload_sql_count**, **global_workload_sql_count**, **gs_workload_sql_elapse_time**, **pgxc_workload_sql_elapse_time**, or **global_workload_sql_elapse_time** are queried.
  - If both **track_activities** and **track_sql_count** are set to **off**, two logs indicating that **track_activities** is disabled and **track_sql_count** is disabled will be displayed when the views are queried.
  - If **track_activities** is set to **off** and **track_sql_count** is set to **on**, a log indicating that **track_activities** is disabled will be displayed when the views are queried.
- If this parameter is disabled, querying the view returns **0**.

enable_track_wait_event

Parameter description: Specifies whether to collect statistics on waiting events, including the number of occurrence times, number of failures, duration, maximum waiting time, minimum waiting time, and average waiting time.

Type: SIGHUP

Value range: Boolean

- **on** indicates that the statistics collection function is enabled.
- **off** indicates that the statistics collection function is disabled.

Default value: **off**

NOTE

- The **enable_track_wait_event** parameter is restricted by **track_activities**. Its functions cannot take effect no matter whether it is enabled if **track_activities** is disabled.
- When **track_activities** or **enable_track_wait_event** is disabled, if you query the **get_instr_wait_event** function, **gs_wait_events** view, or **pgxc_wait_events** view, a message is displayed indicating that the GUC parameter is disabled and the query result is 0.
- If **track_activities** or **enable_track_wait_event** is disabled during cluster running, GaussDB(DWS) will not collect statistics on waiting events. However, statistics that have been collected are not affected.

enable_wdr_snapshot

Parameter description: Specifies whether to enable the performance view snapshot function. After this function is enabled, GaussDB(DWS) will periodically create snapshots for some system performance views and save them permanently. In addition, it will accept manual snapshot creation requests.

Type: SIGHUP

Value range: Boolean

- **on** indicates that the snapshot function is enabled.
- **off** indicates that the snapshot function is disabled.

Default value: **off**

NOTE

- If the **create_wdr_snapshot** function is executed to manually create a view when the **enable_wdr_snapshot** parameter is disabled, a message is displayed indicating that the GUC parameter is not enabled.
- If the **enable_wdr_snapshot** parameter is modified during the snapshot creation process, the snapshot that is being created is not affected. The modification takes effect when the snapshot is manually or periodically created next time.

wdr_snapshot_interval

Parameter description: Specifies the interval for automatically creating performance view snapshots.

Type: SIGHUP

Value range: an integer ranging from 10 to 180, in minutes

Default value: **60**

NOTE

- The value of this parameter must be set in accordance with the cluster load. You are advised to set this parameter to a value greater than the time required for creating a snapshot.
- If the value of **wdr_snapshot_interval** is less than the time required for creating a snapshot, the system will skip this snapshot creation because it finds that the previous snapshot creation is not complete when the time for this automatic snapshot creation arrives.

wdr_snapshot_retention_days

Parameter description: Specifies the maximum number of days for storing performance snapshot data.

Type: SIGHUP

Value range: an integer ranging from 1 to 15 days

Default value: **8**

NOTE

- If **enable_wdr_snapshot** is enabled, snapshot data that has been stored for **wdr_snapshot_retention_days** days will be automatically deleted.
- The value of this parameter must be set in accordance with the available disk space. A larger value requires more disk space.
- The modification of this parameter does not take effect immediately. The expired snapshot data will be cleared only when a snapshot is automatically created next time.

15.12.2 Performance Statistics

During the running of the database, the lock access, disk I/O operation, and invalid message process are involved. All these operations are the bottleneck of the database performance. The performance statistics method provided by GaussDB(DWS) can facilitate the performance fault location.

Generating Performance Statistics Logs

Parameter description: For each query, the following four parameters control the performance statistics of corresponding modules recorded in the server log:

- The **og_parser_stats** parameter controls the performance statistics of a parser recorded in the server log.
- The **log_planner_stats** parameter controls the performance statistics of a query optimizer recorded in the server log.
- The **log_executor_stats** parameter controls the performance statistics of an executor recorded in the server log.
- The **log_statement_stats** parameter controls the performance statistics of the whole statement recorded in the server log.

All these parameters can only provide assistant analysis for administrators, which are similar to the getrusage() of the Linux OS.

Type: SUSET

NOTICE

- **log_statement_stats** records the total statement statistics while other parameters only record statistics about each statement.
- The **log_statement_stats** parameter cannot be enabled together with other parameters recording statistics about each statement.

Value range: Boolean

- **on** indicates the function of recording performance statistics is enabled.
- **off** indicates the function of recording performance statistics is disabled.

Default value: **off**

15.13 Resource Management

If database resource usage is not controlled, concurrent tasks easily 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(DWS) workload management function balances the database workload based on available resources to avoid database overloading.

use_workload_manager

Parameter description: Specifies whether to enable the resource management function. This parameter must be applied on both CNs and DNs.

Type: SIGHUP

Value range: Boolean

- **on** indicates the resource management function is enabled.
- **off** indicates the resource management function is disabled.

NOTE

- If method 2 in **Setting GUC Parameters** 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**, the resource management view becomes available, and you can query the storage resource usage collected in the **off** state. If there are slight errors and the storage resource usage needs to be corrected, run the following command. If data is inserted into the table during the command execution, the statistics may be inaccurate.

```
SELECT gs_wlm_readjust_user_space(0);
```

Default value: **on**

enable_perm_space

Parameter description: Specifies whether to enable the perm space function. This parameter must be applied on both CNs and DNs.

Type: POSTMASTER

Value range: Boolean

- **on** indicates the perm space function is enabled.
- **off** indicates the perm space function is disabled.

Default value: **on**

space_once_adjust_num

Parameter description: In the space control and space statistics functions, specifies the threshold of the number of files processed each time during slow building and fine-grained calibration. This parameter is supported by version 8.1.3 or later clusters.

Type: SIGHUP

Value range: an integer ranging from 1 to INT_MAX

- The value **0** indicates that the slow build and fine-grained calibration functions are disabled.

Default value: **300**

NOTE

The file quantity threshold affects database resources. You are advised to set the threshold to a proper value.

space_readjust_schedule

Parameter description: In the space control and space statistics functions, specifies the space error threshold for triggering automatic calibration. This parameter is supported by version 8.1.3 or later clusters.

Type: SIGHUP

Value range: string

- **off** indicates that the automatic calibration function is disabled.
- **auto** indicates that the automatic calibration function is enabled and the error threshold for triggering automatic calibration is **1 GB**.
- **auto (*space size* + K/M/G)** indicates that the automatic calibration is enabled and the error threshold for triggering automatic calibration is *xxx* KB/MB/GB (user-defined). For example, **auto(200M)** indicates that the automatic calibration is enabled and the error threshold for triggering automatic calibration is **200 MB**.

Default value: **auto**

max_active_statements

Parameter description: Specifies the maximum global concurrency. This parameter applies to one CN.

The database administrator changes the value of this parameter based on system resources (for example, CPU, I/O, and memory resources) so that the system fully supports the concurrency tasks and avoids too many concurrency tasks resulting in system crash.

Type: SIGHUP

Value range: an integer ranging from –1 to INT_MAX. The values **–1** and **0** indicate that the number of concurrent requests is not limited.

Default value: **60**

parctl_min_cost

Parameter description: Specifies the minimum estimated cost of a complex job under static resource management. This parameter sets the threshold for categorizing jobs as simple or complex. Jobs with a cost estimate lower than this value are considered simple, while those with a cost estimate equal to or higher than this value are considered complex.

Type: SIGHUP

Value range: an integer ranging from –1 to INT_MAX

- If **parctl_min_cost** is **-1**, all jobs are simple jobs.
- Jobs whose estimated cost is less than 10 are simple jobs.

Default value: **100000**

cgroup_name

Parameter description: Specifies the name of the Cgroup in use. It can be used to change the priorities of jobs in the queue of a Cgroup.

If you set **cgroup_name** and then **session_respool**, the Cgroups associated with **session_respool** take effect. If you reverse the order, Cgroups 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.

Type: USERSET

You are not advised to set **cgroup_name** and **session_respool** at the same time.

Value range: a string

Default value: **DefaultClass:Medium**

NOTE

DefaultClass:Medium indicates the **Medium** Cgroup belonging to the **Timeshare** Cgroup under the **DefaultClass** Cgroup.

cpu_collect_timer

Parameter description: Specifies how frequently CPU data is collected during statement execution on DNs.

The database administrator changes the value of this parameter based on system resources (for example, CPU, I/O, and memory resources) so that the system fully supports the concurrency tasks and avoids too many concurrency tasks resulting in system crash.

Type: SIGHUP

Value range: an integer ranging from 1 to INT_MAX. The unit is second.

Default value: **30**

enable_cgroup_switch

Parameter description: Specifies whether the database automatically switches to the **TopWD** group when executing statements by group type.

Type: USERSET

Value range: Boolean

- **on**: The database automatically switches to the **TopWD** group when executing statements by group type.
- **off**: The database does not automatically switch to the **TopWD** group when executing statements by group type.

Default value: **off**

memory_tracking_mode

Parameter description: Specifies the memory information recording mode.

Type: USERSET

Value range:

- **none**: Memory statistics is not collected.
- **normal:** Only memory statistics is collected in real time and no file is generated.

- **executor:** The statistics file is generated, containing the context information about all allocated memory used by the execution layer.
- **fullexec**: The generated file includes the information about all memory contexts requested by the execution layer.

Default value: **none**

memory_detail_tracking

Parameter description: Specifies the sequence number of the memory background information distributed in the needed thread and **plannodeid** of the query where the current thread is located.

Type: USERSET

Value range: a string

Default value: empty

NOTICE

It is recommended that you retain the default value for this parameter.

enable_resource_track

Parameter description: Specifies whether the real-time resource monitoring function is enabled. This parameter must be applied on both CNs and DNs.

Type: SIGHUP

Value range: Boolean

- **on** indicates the resource monitoring function is enabled.
- **off** indicates the resource monitoring function is disabled.

Default value: **on**

enable_resource_record

Parameter description: Specifies whether resource monitoring records are archived. When this parameter is enabled, records that have been executed are archived to the corresponding **INFO** views (**GS_WLM_SESSION_INFO** and **GS_WLM_OPERAROR_INFO**). This parameter must be applied on both CNs and DNs.

Type: SIGHUP

Value range: Boolean

- **on** indicates that the resource monitoring records are archived.
- **off** indicates that the resource monitoring records are not archived.

Default value: **on**

NOTE

The default value of this parameter is **on** for a new cluster. In upgrade scenarios, the default value of this parameter is the same as that of the source version.

enable_track_record_subsql

Parameter description: Specifies whether to enable the function of recording and archiving sub-statements. When this function is enabled, sub-statements in stored procedures and anonymous blocks are recorded and archived to the corresponding **INFO** table (**GS_WLM_SESSION_INFO**). This parameter is a session-level parameter. It can be configured and take effect in the session connected to the CN and affects only the statements in the session. It can also be configured on both the CN and DN and take effect globally.

Type: USERSET

Value range: Boolean

- **on** indicates that the sub-statement resource monitoring records are archived.
- **off** indicates that the sub-statement resource monitoring records are not archived.

Default value: **off**

enable_user_metric_persistent

Parameter description: Specifies whether the user historical resource monitoring dumping function is enabled. When this function is enabled, data in the **PG_TOTAL_USER_RESOURCE_INFO** view is periodically sampled and saved to the **GS_WLM_USER_RESOURCE_HISTORY** system catalog, and data in the **GS_RESPOOL_RESOURCE_INFO** view is periodically sampled and saved to the **GS_RESPOOL_RESOURCE_HISTORY** system catalog.

Type: SIGHUP

Value range: Boolean

- **on** indicates that the user historical resource monitoring dumping function is enabled.
- **off** indicates that the user historical resource monitoring dumping function is disabled.

Default value: **on**

user_metric_retention_time

Parameter description: Specifies the retention time of the user historical resource monitoring data. This parameter is valid only when **enable_user_metric_persistent** is set to **on**.

Type: SIGHUP

Value range: an integer ranging from 0 to 3650. The unit is day.

- If this parameter is set to **0**, user historical resource monitoring data is permanently stored.

- If the value is greater than **0**, user historical resource monitoring data is stored for the specified number of days.

Default value: 7

enable_instance_metric_persistent

Parameter description: Specifies whether the instance resource monitoring dumping function is enabled. When this function is enabled, the instance monitoring data is saved to the system catalog **GS_WLM_INSTANCE_HISTORY**.

Type: SIGHUP

Value range: Boolean

- **on** indicates that the instance resource monitoring dumping function is enabled.
- **off**: Specifies that the instance resource monitoring dumping function is disabled.

Default value: **on**

instance_metric_retention_time

Parameter description: Specifies the retention time of the instance historical resource monitoring data. This parameter is valid only when **enable_instance_metric_persistent** is set to **on**.

Type: SIGHUP

Value range: an integer ranging from 0 to 3650. The unit is day.

- If this parameter is set to **0**, instance historical resource monitoring data is permanently stored.
- If the value is greater than **0**, the instance historical resource monitoring data 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**.

Type: USERSET

Value range: enumerated values

- **none**: Resources are not monitored.
- **query**: Enables query-level resource monitoring. If this function is enabled, the plan information (similar to the output information of EXPLAIN) of SQL statements will be recorded in top SQL statements.
- **perf**: Enables the perf-level resource monitoring. If this function is enabled, the plan information (similar to the output information of EXPLAIN ANALYZE) that contains the actual execution time and the number of execution rows will be recorded in the top SQL.

- **operator**: enables the operator-level resource monitoring. If this function is enabled, not only the information including the actual execution time and number of execution rows is recorded in the top SQL statement, but also the operator-level execution information is updated to the top SQL statement.

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**.

Type: USERSET

Value range: an integer ranging from –1 to INT_MAX

- **–1** indicates that resource monitoring is disabled.
- A value greater than or equal to **0** indicates that statements whose execution cost exceeds this value will be monitored.

Default value: **0**

NOTE

The default value of this parameter is **0** for a new cluster. In upgrade scenarios, the default value of this parameter is the same as that of the source version.

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 (see **Table 12-1**) 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**.

Type: USERSET

Value range: an integer ranging from 0 to INT_MAX. The unit is second (s).

- **0** indicates that information about all statements recorded in the real-time resource monitoring view (see **Table 12-1**) will be archived into historical views.
- When the value is greater than **0**, the system archives historical information if the total execution and queuing time of statements in the real-time resource monitoring view (**Table 12-1**) goes over the parameter value.

Default value: **60s**

dynamic_memory_quota

Parameter description: Specifies the memory quota in adaptive load scenarios, that is, the proportion of maximum available memory to total system memory.

Type: SIGHUP

Value range: an integer ranging from 1 to 100

Default value: **80**

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.

Type: USERSET

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.

Type: USERSET

Value range: a string

Default value: empty

enable_dynamic_workload

Parameter description: Specifies whether to enable the dynamic workload management function.

Type: POSTMASTER

Value range: Boolean

- **on** indicates the dynamic workload management function is enabled.
- **off** indicates the dynamic workload management function is disabled.

Default value: on

NOTICE

- If memory adaptation is enabled, you do not need to use **work_mem** to optimize the operator memory usage after collecting statistics. The system will generate a plan for each statement based on the current load, estimating the memory used by each operator and by the entire statement. In a concurrency scenario, statements are queued based on the system load and their memory usage.
- The optimizer cannot accurately estimate the number of rows and will probably underestimate or overestimate memory usage. If the memory usage is underestimated, the allocated memory will be automatically increased during statement running. If the memory usage is overestimated, system resources will not be fully used, and the number of statements waiting in a queue will increase, which probably results in low performance. To improve performance, identify the statements whose estimated memory usage is much greater than the DN peak memory and adjust the value of **query_max_mem**. For details, see **Adjusting Key Parameters During SQL Tuning**.

bbox_dump_count

Parameter description: Specifies the maximum number of core files that are generated by GaussDB(DWS) 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 if **enable_bbox_dump** is set to **on**.

Type: USERSET

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**.

io_limits

Parameter description: This parameter has been discarded in version 8.1.2 and is reserved for compatibility with earlier versions. This parameter is invalid in the current version.

Type: USERSET

Value range: an integer ranging from 0 to 1073741823

Default value: **0**

io_priority

Parameter description: This parameter has been discarded in version 8.1.2 and is reserved for compatibility with earlier versions. This parameter is invalid in the current version.

Type: USERSET

Value range: enumerated values

- None
- Low
- Medium
- High

Default value: **None**

session_respool

Parameter description: Specifies the resource pool associated with the current session.

Type: USERSET

If you set **cgroup_name** and then **session_respool**, the Cgroups associated with **session_respool** take effect. If you reverse the order, Cgroups 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 not advised 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**

enable_transaction_parctl

Parameter description: whether to control transaction block statements and stored procedure statements.

Type: USERSET

Value range: Boolean

- **on**: Transaction block statements and stored procedure statements are controlled.
- **off**: Transaction block statements and stored procedure statements are not controlled.

Default value: **on**

session_history_memory

Parameter description: Specifies the memory size of a historical query view.

Type: SIGHUP

Value range: an integer ranging from 10240 to 50% of **max_process_memory**. The unit is KB.

Default value: 100 MB

topsql_retention_time

Parameter description: Specifies the retention period of historical Top SQL data in the **gs_wlm_session_info** and **gs_wlm_operator_info** tables.

Type: SIGHUP

Value range: an integer ranging from 0 to 3650. 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: **30**

CAUTION

- Before setting this GUC parameter to enable the data retention function, delete data from the **gs_wlm_session_info** and **gs_wlm_operator_info** tables.
- The default value of this parameter is **30** for a new cluster. In upgrade scenarios, the default value of this parameter is the same as that of the source version.

---

transaction_pending_time

Parameter description: maximum queuing time of transaction block statements and stored procedure statements if **enable_transaction_parctl** is set to **on**.

Type: USERSET

Value range: an integer ranging from –1 to INT_MAX. The unit is second (s).

- **–1** or **0**: No queuing timeout is specified for transaction block statements and stored procedure statements. The statements can be executed when resources are available.
- Value greater than **0**: 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**

NOTICE

This parameter is valid only for internal statements of stored procedures and transaction blocks. That is, this parameter takes effect only for the statements whose **enqueue** value (for details, see **PG_SESSION_WLMSTAT**) is **Transaction** or **StoredProc**.

---

wlm_sql_allow_list

Parameter description: Specifies whitelisted SQL statements for resource management. Whitelisted SQL statements are not monitored by resource management.

Type: SIGHUP

Value range: a string

Default value: empty

NOTICE

- One or more whitelisted SQL statements can be specified in **wlm_sql_allow_list**. If multiple SQL statements are to be whitelisted, use semicolons (;) to separate them.
- The system determines whether SQL statements are monitored based on the prefix match. The SQL statements are case insensitive. For example, if **wlm_sql_allow_list** is set to **'SELECT'**, all **SELECT** statements are not monitored by the resource management module.
- The system identifies spaces at the beginning of the parameter value. For example, **'SELECT'** and **' SELECT'** have different representations. **' SELECT'** filters only the **SELECT** statements with spaces at the beginning.
- The system has some whitelisted SQL statements by default, which cannot be modified. You can query the default whitelisted SQL statements and the SQL statements that have been successfully added to the whitelist by GUC through the system view **gs_wlm_sql_allow**.
- New SQL statements cannot be appended to the whitelisted SQL statements specified by **wlm_sql_allow_list** but can be set only through overwriting. To add an SQL statement, query the original GUC value, add the new statement to the end of the original value, separate the statements with a semicolon (;), and set the GUC value again.

15.14 Automatic Cleanup

The automatic cleanup process (**autovacuum**) in the system automatically runs the **VACUUM** and **ANALYZE** statements to reclaim the record space marked as deleted and update statistics about the table.

autovacuum

Parameter description: Specifies whether to start the automatic cleanup process (**autovacuum**). Ensure that the **track_counts** parameter is set to **on** before enabling the automatic cleanup process.

For clusters of 8.1.3 or later, the automatic cleanup function can be performed on the management console. For details, see "Intelligent O&M Overview" in the Data Warehouse Service User Guide. For clusters of 8.1.2 or earlier, configure GUC parameters according to **Configuring GUC Parameters**.

Type: SIGHUP

Value range: Boolean

- **on** indicates the database automatic cleanup process is enabled.
- **off** indicates that the database automatic cleanup process is disabled.

Default value: **on**

NOTE

Set **autovacuum** to **on** if you want to enable the function of automatically cleaning up two-phase transactions after the system recovers from faults.

- If **autovacuum** is set to **on** and **autovacuum_max_workers** to **0**, the **autovacuum** process will not be automatically performed and only abnormal two-phase transactions are cleaned up after the system recovers from faults.
- If **autovacuum** is set to **on** and the value of **autovacuum_max_workers** is greater than **0**, the system will automatically clean up two-phase transactions and processes after recovering from faults.

NOTICE

Even if this parameter is set to **off**, the database initiates a cleanup process when transaction ID wraparound needs to be prevented. When a **CREATE DATABASE** or **DROP DATABASE** operation fails, the transaction may have 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:

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 whose transactions are in the prepared status. Administrators connect to an available database such as gaussdb to run the **SET xc_maintenance_mode = on** statement.
3. Commit or roll back the two-phase transaction based on the global transaction status.

autovacuum_mode

Parameter description: Specifies whether the **autoanalyze** or **autovacuum** function is enabled. This parameter is valid only when **autovacuum** is set to **on**.

Type: SIGHUP

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 **analyze** on a table exceeds the value of **autoanalyze_timeout**, **analyze** is automatically canceled.

Type: SIGHUP

Value range: an integer ranging from 0 to 2147483. The unit is second.

Default value: **5min**

autovacuum_io_limits

Parameter description: Specifies the upper limit of I/Os triggered by the **autovacuum** process per second. This parameter has been discarded in version 8.1.2 and is reserved for compatibility with earlier versions. This parameter is invalid in the current version.

Type: SIGHUP

Value range: an integer ranging from –1 to 1073741823. **–1** indicates that the default Cgroup is used.

Default value: **–1**

autovacuum_max_workers

Parameter description: Specifies the maximum number of automatic cleanup threads running at the same time.

Type: SIGHUP

Value range: an integer ranging from 0 to 128. **0** indicates that **autovacuum** is disabled.

Default value: **3**

NOTE

- This parameter works with **autovacuum**. The rules for clearing system catalogs and user tables are as follows:
  - When **autovacuum_max_workers** is set to **0**, **autovacuum** is disabled and no tables are cleared.
  - If **autovacuum_max_workers > 0** and **autovacuum = off** are configured, the system only clears the system catalogs and column-store tables with delta tables enabled (such as **vacuum delta tables**, **vacuum cudesc tables**, and **delta merge**).
  - When **autovacuum_max_workers** is set to a value greater than zero and **autovacuum** is enabled, all tables will be cleared.
- In 8.1.3, column-store primary tables are not cleared by default. You need to set the **colvacuum_threshold_scale_factor** parameter to enable this function.

autovacuum_naptime

Parameter description: Specifies the interval between two automatic cleanup operations.

Type: SIGHUP

Value range: an integer ranging from 1 to 2147483. The unit is second.

Default value: **60s**

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.

Type: SIGHUP

Value range: an integer ranging from **0** to **INT_MAX**

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.

Type: SIGHUP

Value range: an integer ranging from **0** to **INT_MAX**

Default value:

- If the current cluster is upgraded from an earlier version to 8.1.3, the default value is **10000** to ensure forward compatibility.
- If the current cluster version is 8.1.3, the default value is **50**.

autovacuum_vacuum_scale_factor

Parameter description: Specifies the size scaling factor of a table added to the **autovacuum_vacuum_threshold** parameter when a **VACUUM** event is triggered.

Type: SIGHUP

Value range: a floating point number ranging from 0.0 to 100.0

Default value: **0.2**

autovacuum_analyze_scale_factor

Parameter description: Specifies the size scaling factor of a table added to the **autovacuum_analyze_threshold** parameter when an **ANALYZE** event is triggered.

Type: SIGHUP

Value range: a floating point number ranging from 0.0 to 100.0

Default value:

- If the current cluster is upgraded from an earlier version to 8.1.3, the default value is **0.25** to ensure forward compatibility.
- If the current cluster version is 8.1.3, the default value is **0.1**.

autovacuum_freeze_max_age

Parameter description: Specifies the maximum age (in transactions) that a table's **pg_class.relfrozenxid** column can attain before a VACUUM operation is forced to prevent transaction ID wraparound within the table.

The old files under the subdirectory of **pg_clog/** can also be deleted by the VACUUM operation. Even if the automatic cleanup process is forbidden, the system will invoke the automatic cleanup process to prevent the cyclic repetition.

Type: SIGHUP

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.

Type: SIGHUP

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: **2ms**

autovacuum_vacuum_cost_limit

Parameter description: Specifies the value of the cost limit used in the **autovacuum** operation.

Type: SIGHUP

Value range: an integer ranging from –1 to 10000. **-1** indicates that the normal vacuum cost limit is used.

Default value: **–1**

15.15 Default Settings of Client Connection

15.15.1 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 (,).

Type: USERSET

- If the schema of a temporary table exists in the current session, the scheme can be listed in **search_path** by using the alias **pg_temp**, for example, **'pg_temp,public'**. The schema of a temporary table has the highest search priority and is always searched before all the 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 only searched for database objects, including tables, views, and data types. Functions or operator names will not be searched for.
- The schema of a system catalog, **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 specifying a particular schema, the object will be placed in the first valid schema listed in **search_path**. An error will be 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**, a database can be shared (where no users have private schemas, and all share use of public), and 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.

Type: USERSET

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 specifies 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 consulted instead.
- This parameter is not used when users create databases. By default, a new database inherits its tablespace setting from the template database.

Type: USERSET

Value range: a string. An empty string indicates that the default tablespace is used.

Default value: empty

default_storage_nodegroup

Parameter description: Specifies the Node Group where a table is created by default. This parameter takes effect only for ordinary tables.

Type: USERSET

Value range: a string

- **installation**: indicates that the table is created in the installed Node Group by default.
- **random_node_group**: indicates that the table is created in a randomly selected Node Group by default. This feature is supported in 8.1.2 or later and is used only in the test environment.
- **roach_group**: indicates that the table is created in all nodes by default. This value is reserved for the Roach tool and cannot be used in other scenarios.
- A value other than the preceding three options indicates that the table is created in a specified Node Group.

Default value: **installation**

default_colversion

Parameter description: Sets the storage format version of the column-store table that is created by default.

Type: SIGHUP

Value range: enumerated values

- **1.0**: Each column in a column-store table is stored in a separate file. The file name is **relfilenode.C1.0**, **relfilenode.C2.0**, **relfilenode.C3.0**, or similar.
- **2.0**: All columns of a column-store table are combined and stored in a file. The file is named **relfilenode.C1.0**.

Default value: 2.0

temp_tablespaces

Parameter description: Specifies tablespaces to which temporary objects will be created (temporary tables and their indexes) when a **CREATE** command does not explicitly specify a tablespace. Temporary files for sorting large data are created in these tablespaces.

The value of this parameter is a list of names of tablespaces. When there is more than one name in the list, GaussDB(DWS) chooses a random tablespace from the list upon the creation of a temporary object each time. Except that 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(DWS) will automatically use the default tablespace of the current database instead.

Type: USERSET

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.

Type: USERSET

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.

Type: USERSET

Value range: enumerated values

- **READ COMMITTED**: Only committed data is read. This is the default.
- **READ UNCOMMITTED**: GaussDB(DWS) does not support **READ UNCOMMITTED**. If **READ UNCOMMITTED** is set, **READ COMMITTED** is executed instead.

- **REPEATABLE READ**: Only the data committed before transaction start is read. Uncommitted data or data committed in other concurrent transactions cannot be read.
- **SERIALIZABLE**: GaussDB(DWS) does not support **SERIALIZABLE**. If **SERIALIZABLE** is set, **REPEATABLE READ** is executed instead.

Default value: **READ COMMITTED**

default_transaction_read_only

Parameter description: Specifies whether each new transaction is in read-only state.

Type: SIGHUP

Value range: Boolean

- **on** indicates the transaction is in read-only state.
- **off** indicates the transaction is in read/write state.

Default value: **off**

default_transaction_deferrable

Parameter description: Specifies the default delaying state of each new transaction. It currently has no effect on read-only transactions or those running at isolation levels lower than serializable.

GaussDB(DWS) does not support the serializable isolation level of each transaction. The parameter is insignificant.

Type: USERSET

Value range: Boolean

- **on** indicates a transaction is delayed by default.
- **off** indicates 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.

Type: USERSET

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 when the server receives the command) is longer than the duration specified by the parameter, error information is displayed when you attempt to execute the statement and the statement then exits.

Type: USERSET

Value range: an integer ranging from 0 to 2147483647. The unit is ms.

Default value:

- If the current cluster is upgraded from an earlier version to 8.1.3, the value in the earlier version is inherited. The default value is **0**.
- If the current cluster version is 8.1.3, the default value is **24h**.

vacuum_freeze_min_age

Parameter description: Specifies the minimum cutoff age (in the same transaction), based on which **VACUUM** decides whether to replace transaction IDs with FrozenXID while scanning a table.

Type: USERSET

Value range: an integer from 0 to 576460752303423487.

NOTE

Although you can set this parameter to a value ranging from **0** to **1000000000** anytime, **VACUUM** will limit the effective value to half the value of autovacuum_freeze_max_age by default.

Default value: **5000000000**

vacuum_freeze_table_age

Parameter description: Specifies the time that VACUUM freezes tuples while scanning the whole table. **VACUUM** performs a whole-table scan if the value of the **pg_class.relfrozenxid** column of the table has reached the specified time.

Type: USERSET

Value range: an integer from 0 to 576460752303423487.

NOTE

Although users can set this parameter to a value ranging from **0** to **2000000000** anytime, **VACUUM** will limit the effective value to 95% of autovacuum_freeze_max_age by default. Therefore, a periodic manual VACUUM has a chance to run before an anti-wraparound autovacuum is launched for the table.

Default value: **15000000000**

bytea_output

Parameter description: Specifies the output format for values of the bytea type.

Type: USERSET

Value range: enumerated values

- **hex** indicates the binary data is converted to the two-byte hexadecimal digit.
- **escape** indicates 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.

Type: USERSET

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.

Type: USERSET

Value range: enumerated values

- **document** indicates an HTML document.
- **content** indicates a common string.

Default value: **content**

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 modifying index storage parameters.

Type: USERSET

Value range: an integer ranging from 64 to INT_MAX. The unit is KB.

Default value: **4 MB**

15.15.2 Zone 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.

Type: USERSET

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**.

Suggestion: The ISO format is recommended. Postgres, SQL, and German use abbreviations for time zones, such as **EST**, **WST**, and **CST**.

IntervalStyle

Parameter description: Specifies the display format for interval values.

Type: USERSET

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.
- **oracle** indicates the output result that matches the numtodsinterval function in the Oracle database. 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 time stamps.

Type: USERSET

Value range: a string. You can obtain it by querying the **pg_timezone_names** view.

Default value: **UTC**

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.

Type: USERSET

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. There are also other abbreviations, such as **Australia** and **India** that can be defined for a particular installation.

extra_float_digits

Parameter description: Specifies 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).

Type: USERSET

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 type (character set).

Set this parameter as needed. Try to keep the client code and server code consistent to improve efficiency.

Type: USERSET

Value range: encoding compatible with PostgreSQL. **UTF8** indicates that the database encoding is used.

NOTE

- You can run the **locale -a** command to check and set the system-supported zone and the corresponding encoding format.
- 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 a cluster, 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.

Valid values depend on the current system. On some systems, this zone 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.

Type: SUSET

Value range: a string

NOTE

- You can run the **locale -a** command to check and set the system-supported zone and the corresponding encoding format.
- 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. Valid values depend on the current system.

Type: USERSET

Value range: a string

NOTE

- You can run the **locale -a** command to check and set the system-supported zone and the corresponding encoding format.
- 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. Valid values depend on the current system.

Type: USERSET

Value range: a string

NOTE

- You can run the **locale -a** command to check and set the system-supported zone and the corresponding encoding format.
- 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 zones. It affects the output of functions such as to_char. Valid values depend on the current system.

Type: USERSET

Value range: a string

NOTE

- You can run the **locale -a** command to check and set the system-supported zone and the corresponding encoding format.
- 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.

Type: USERSET

Value range: a string

NOTE

GaussDB(DWS) supports the following two configurations: pg_catalog.english and pg_catalog.simple.

Default value: **pg_catalog.english**

15.15.3 Other Default Parameters

This section describes the default database loading parameters of the database system.

dynamic_library_path

Parameter description: Specifies the path for saving the shared database files that are dynamically loaded for data searching. When a dynamically loaded 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.

The value of **dynamic_library_path** must be a list of absolute paths separated by colons (:) or by semi-colons (;) on the Windows OS. The special variable **$libdir** in the beginning of a path will be replaced with the module installation directory provided by GaussDB(DWS). Example:

```
dynamic_library_path = '/usr/local/lib/postgresql:/opt/testgs/lib:$libdir'
```

Type: SUSET

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.

Type: USERSET

Value range: an integer ranging from 0 to INT_MAX. The value **0** indicates no limit.

Default value: **0**

15.16 Lock Management

In GaussDB(DWS), a deadlock may occur when concurrently executed transactions compete for resources. This section describes parameters used for managing transaction lock mechanisms.

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.

- 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 released before the waiter decides to check for deadlocks will be reduced.
- When **log_lock_waits** is set, this parameter also determines the duration you need to wait before a log message about the lock wait is issued. If you are trying to investigate locking delays, you need to set this parameter to a value smaller than normal **deadlock_timeout**.

Type: SUSET

Value range: an integer ranging from 1 to 2147483647. The unit is millisecond (ms).

Default value: **1s**

ddl_lock_timeout

Parameter description: Indicates the number of seconds a DDL command should wait for the locks to become available. If the time spent in waiting for a lock exceeds the specified time, an error is reported. (This parameter is supported only in 8.1.3.200 and later cluster versions.)

Type: SUSET

Value range: an integer ranging from 0 to INT_MAX. The unit is millisecond (ms).

- If the value of this parameter is 0, this parameter does not take effect.
- If the value of this parameter is greater than 0, the lock wait time of DDL statements is the value of this parameter, and the lock wait time of other locks is the value of **lockwait_timeout**.

Default value: **0**

NOTE

This parameter has a higher priority than **lockwait_timeout** and takes effect only for **AccessExclusiveLock**.

lockwait_timeout

Parameter description: Specifies the longest time to wait before a single lock times out. If the time you wait before acquiring a lock exceeds the specified time, an error is reported.

Type: SUSET

Value range: an integer ranging from 0 to INT_MAX. The unit is millisecond (ms).

Default value: **20 min**

update_lockwait_timeout

Parameter description: sets 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 you wait before acquiring a lock exceeds the specified time, an error is reported.

Type: SUSET

Value range: an integer ranging from 0 to INT_MAX. The unit is millisecond (ms).

Default value: **2 min**

max_locks_per_transaction

Parameter description: Controls 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 that do not exceed the preset number can be locked simultaneously at any time. You may need to increase this value when you modify many different tables in a single transaction. This parameter can only be set at database start.
- If this parameter is set to a large value, GaussDB(DWS) may require more System V shared memory than the default setting.
- 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.

Type: POSTMASTER

Value range: an integer ranging from 10 to INT_MAX

Default value: **256**

ddl_select_concurrent_mode

Parameter description: Specifies the concurrency mode of DDL and **SELECT** statements. This parameter is available only in 8.1.3.320 and later cluster versions.

Type: SUSET

Value range: enumerated values

- **none**: DDL and **SELECT** statements cannot be executed concurrently. Waiting statements are in the lock wait state.
- **truncate**: When a **TRUNCATE** statement is blocked by a **SELECT** statement, the **TRUNCATE** statement interrupts the **SELECT** statement and is executed first. Other DDL statements and **SELECT** statements remain in the lock wait state.
- **exchange**: When an **EXCHANGE** statement is blocked by a **SELECT** statement, the **EXCHANGE** statement interrupts the **SELECT** statement and is executed first. Other DDL statements and **SELECT** statements remain in the lock wait state.
- **truncate**, **exchange**: When a **TRUNCATE** and an **EXCHANGE** statement are blocked by the **SELECT** statement, the **SELECT** statement is interrupted and the **TRUNCATE** and **EXCHANGE** statement are executed first.

Default value: **none**

NOTE

- To reserve time for the **SELECT** statement to respond to signals, if the value of **ddl_lock_timeout** is less than 1 second in the current version, 1 second is used.
- Concurrency is not supported when there are conflicts with locks of higher levels (more than one level). For example, **autoanalyze** is triggered by **select** when **autoanalyze_mode** is set to **normal**.
- Concurrency is not supported when there are conflicts with locks in transaction blocks.

max_pred_locks_per_transaction

Parameter description: Controls the average number of predicated locks allocated for each transaction.

- The size of the shared and predicated 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 that do not exceed the preset number can be locked simultaneously at any time. You may need to increase this value when you modify many different tables in a single transaction. This parameter can only be set at server start.
- If this parameter is set to a large value, GaussDB(DWS) may require more System V shared memory than the default setting.

Type: POSTMASTER

Value range: an integer ranging from 10 to INT_MAX

Default value: **64**

partition_lock_upgrade_timeout

Parameter description: Specifies the time to wait before the attempt of a lock upgrade from ExclusiveLock to AccessExclusiveLock times out on partitions.

- 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, ExclusiveLock is acquired for the partitions during data rearrangement and AccessExclusiveLock is acquired during file exchange.

- Generally, a partition waits until it acquires a lock, or a timeout occurs if the partition waits for a period of time longer than specified by the **lockwait_timeout** parameter.
- When doing MERGE PARTITION or CLUSTER PARTITION on a partitioned table, you need to acquire AccessExclusiveLock during file exchange. If the lock fails to be acquired, the acquisition is retried in 50 ms. This 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.

Type: USERSET

Value range: an integer ranging from -1 to 3000. The unit is second (s).

Default value: **1800**

15.17 Version and Platform Compatibility

15.17.1 Compatibility with Earlier Versions

This section describes the parameter control of the downward compatibility and external compatibility features of GaussDB(DWS). Backward compatibility of the database system provides support for the application of databases of earlier versions. This section describes parameters used for controlling backward compatibility of a database.

array_nulls

Parameter description: Determines whether the array input parser recognizes unquoted NULL as a null array element.

Type: USERSET

Value range: Boolean

- **on** indicates that null values can be entered in arrays.
- **off** indicates backward compatibility with the old behavior. Arrays containing **NULL** values can still be created when this parameter is set to **off**.

Default value: **on**

backslash_quote

Parameter description: Determines whether a single quotation mark can be represented by \' in a string text.

Type: USERSET

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**

default_with_oids

Parameter description: Determines 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.

Type: USERSET

Value range: Boolean

- **on** indicates **CREATE TABLE** and **CREATE TABLE AS** can include an **OID** field in newly-created tables.
- **off** indicates **CREATE TABLE** and **CREATE TABLE AS** cannot include any **OID** field in newly-created tables.

Default value: **off**

escape_string_warning

Parameter description: Specifies a warning on directly using a backslash (\) 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 is now to treat the backslash as an ordinary character in each SQL standard.
- This variable can be enabled to help locate codes that need to be changed.

Type: USERSET

Value range: Boolean

Default value: **on**

lo_compat_privileges

Parameter description: Determines whether to enable backward compatibility for the privilege check of large objects.

Type: SUSET

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: When the database generates SQL, this parameter forcibly quotes all identifiers even if they are not keywords. This will affect the output of EXPLAIN as well as the results of functions, such as pg_get_viewdef. For details, see the **--quote-all-identifiers** parameter of **gs_dump**.

Type: USERSET

Value range: Boolean

- **on** indicates the forcible quotation function is enabled.
- **off** indicates the forcible quotation function is disabled.

Default value: **off**

sql_inheritance

Parameter description: Determines whether to inherit semantics.

Type: USERSET

Value range: Boolean

off indicates that child tables cannot be accessed by various commands. That is, an ONLY keyword is used by default. This setting is compatible with versions earlier than PostgreSQL 7.1.

Default value: **on**

standard_conforming_strings

Parameter description: Determines 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'...').

Type: USERSET

Value range: Boolean

- **on** indicates that the function is enabled.
- **off** indicates that the function is disabled.

Default value: **on**

synchronize_seqscans

Parameter description: Controls sequential scans of tables to synchronize with each other. Concurrent scans read the same data block about at the same time and share the I/O workload.

Type: USERSET

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: Controls whether certain limited features, such as GDS table join, are available. These features are not explicitly prohibited in earlier versions, but are not recommended due to their limitations in certain scenarios.

Type: USERSET

Value range: Boolean

- **on** indicates that the features are enabled and forward compatible, but may incur errors in certain scenarios.
- **off** indicates that the features are disabled.

Default value: **off**

15.17.2 Platform and Client Compatibility

Many platforms use the database system. External compatibility of the database system provides a lot of conveniences for platforms.

transform_null_equals

Parameter description: Determines whether expressions of the form expr = NULL (or NULL = expr) are treated as expr IS NULL. They return true if expr evaluates to **NULL**, and false otherwise.

- The correct SQL-standard-compliant behavior of expr = NULL is to always return null (unknown).
- Filtered forms in MS 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.

Type: USERSET

Value range: Boolean

- **on** indicates expressions of the form expr = NULL (or NULL = expr) are treated as expr IS NULL.
- **off** indicates expr = NULL always returns NULL.

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.

td_compatible_truncation

Parameter description: Determines 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 (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.

Type: USERSET

Value range: Boolean

- **on** indicates overlong strings are truncated.
- **off** indicates overlong strings are not truncated.

Default value: **off**

15.18 Fault Tolerance

This section describes parameters used for controlling the methods that the server processes an error occurring in the database system.

exit_on_error

Parameter description: Specifies whether to terminate the current session.

Type: SUSET

Value range: Boolean

- **on** indicates that any error will terminate the current session.
- **off** indicates that only a FATAL error will terminate the current session.

Default value: **off**

omit_encoding_error

Parameter description: This parameter determines how to handle character code errors that occur when converting a database to UTF-8. If set to true, it replaces the invalid characters with question marks (?).

Type: USERSET

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

Parameter description: Specifies the maximum number of automatic retry times when an SQL statement error occurs. Currently, a statement can start retrying if the following errors occur: **Connection reset by peer**, **Lock wait timeout**, and **Connection timed out**. If this parameter is set to **0**, the retry function is disabled.

Type: USERSET

Value range: an integer ranging from 0 to 20

Default value: **6**

max_cn_temp_file_size

Parameter description: Specifies the maximum number of temporary files that can be used by the CN during automatic SQL statement retries. The value **0** indicates that no temporary file is used.

Type: SIGHUP

Value range: an integer ranging from 0 to 10485760. The unit is KB.

Default value: **5 GB**

retry_ecode_list

Parameter description: Specifies the list of SQL error types that support automatic retry.

Type: USERSET

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 CG003 CG004 F0011 45003

15.19 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.

min_pool_size

Parameter description: Specifies the minimum number of connections between a CN's connection pool and another CN/DN.

Type: POSTMASTER

Value range: an integer ranging from 1 to 65535

Default value: **1**

max_pool_size

Parameter description: Specifies the maximum number of connections between a CN's connection pool and another CN/DN.

Type: POSTMASTER

Value range: an integer ranging from 1 to 65535

Default value: **800** for CNs and **5000** for DNs

persistent_datanode_connections

Parameter description: Specifies whether to release the connection for the current session.

Type: USERSET

Value range: Boolean

- **off** indicates that the connection for the current session will be released.
- **on** indicates that the connection for the current session will not be released.

NOTICE

After this function is enabled, a session may hold a connection but does not run a query. As a result, other query requests fail to be connected. To fix this problem, the number of sessions must be less than or equal to **max_active_statements**.

Default value: **off**

cache_connection

Parameter description: Specifies whether to reclaim the connections of a connection pool.

Type: USERSET

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**

enable_force_reuse_connections

Parameter description: Specifies whether a session forcibly reuses a new connection.

Type: USERSET

Value range: Boolean

- **on** indicates that the new connection is forcibly used.
- **off** indicates that the current connection is used.

Default value: **off**

NOTE

This is a session connection parameter. You are advised not to configure this parameter.

enable_pooler_parallel

Parameter description: Specifies whether a CN's connection pool can be connected in parallel mode.

Type: SIGHUP

Value range: Boolean

- **on** indicates that a CN's connection pool can be connected in parallel mode.
- **off** indicates that a CN's connection pool cannot be connected in parallel mode.

Default value: **on**

15.20 Cluster Transaction Parameters

This section describes the settings and value ranges of cluster transaction parameters.

transaction_isolation

Parameter description: Specifies the isolation level of the current transaction.

Type: USERSET

Value range:

- **READ COMMITTED**: Only committed data is read. This is the default.
- **READ UNCOMMITTED**: GaussDB(DWS) does not support **READ UNCOMMITTED**. If **READ UNCOMMITTED** is set, **READ COMMITTED** is executed instead.
- **REPEATABLE READ**: Only the data committed before transaction start is read. Uncommitted data or data committed in other concurrent transactions cannot be read.
- **SERIALIZABLE**: GaussDB(DWS) does not support **SERIALIZABLE**. If **SERIALIZABLE** is set, **REPEATABLE READ** is executed instead.

Default value: **READ COMMITTED**

transaction_read_only

Parameter description: Specifies that the current transaction is a read-only transaction.

Type: USERSET

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** for CNs and **on** for DNs

xc_maintenance_mode

Parameter description: Specifies whether the system is in maintenance mode.

Type: SUSET

Value range: Boolean

- **on** indicates that maintenance mode is enabled.
- **off** indicates that the maintenance mode is disabled.

Default value: **off**

NOTICE

Enable the maintenance mode with caution to avoid cluster data inconsistencies.

allow_concurrent_tuple_update

Parameter description: Specifies whether to allow concurrent update.

Type: USERSET

Value range: Boolean

- **on** indicates it is enabled.
- **off** indicates it is disabled.

Default value: **on**

gtm_backup_barrier

Parameter description: Specifies whether to create a restoration point for the GTM starting point.

Type: SUSET

Value range: Boolean

- **on** indicates that a restoration point will be created for the GTM starting point.
- **off** indicates that a restoration point will not be created for the GTM starting point.

Default value: **off**

gtm_conn_check_interval

Parameter description: Sets the CN to check whether the connection between the local thread and the primary GTM is normal.

Parameter type: SIGHUP

Value range: an integer ranging from 0 to INT_MAX/1000. The unit is second.

Default value: **10s**

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. Currently, this parameter is not used in GaussDB(DWS). Similar to this parameter, the **default_transaction_deferrable** parameter is used to specify whether to allow delayed execution of a transaction.

Type: USERSET

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**

enforce_two_phase_commit

Parameter description: This parameter is reserved for compatibility with earlier versions. This parameter is invalid in the current version.

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/true**, all versions of tuples in the table are displayed.

Type: USERSET

Value range: Boolean

- **on/true** indicates that all versions of tuples in the table are displayed.
- **off/false** indicates that no versions of tuples in the table are displayed.

Default value: **off**

gtm_connect_retries

Parameter description: Specifies the number of GTM reconnection attempts.

Type: SIGHUP

Value range: an integer ranging from 1 to 2147483647.

Default value: **30**

enable_redistribute

Parameter description: Specifies whether unmatched nodes are redistributed.

Type: SUSET

Value range: Boolean

- **on** indicates that unmatched nodes are redistributed.
- **off** indicates that unmatched nodes are not redistributed.

Default value: **off**

15.21 Developer Operations

enable_light_colupdate

Parameter description: Specifies whether to enable the lightweight column-store update.

Type: USERSET

Value range: Boolean

- **on** indicates that the lightweight column-store update is enabled.
- **off** indicates that the lightweight column-store update is disabled.

Default value: **off**

enable_fast_query_shipping

Parameter description: Specifies whether to use the distributed framework for a query planner.

Type: USERSET

Value range: Boolean

- **on** indicates that execution plans are generated on CNs and DNs separately.
- **off** indicates that the distributed framework is used. Execution plans are generated on CNs and then sent to DNs for execution.

Default value: **on**

enable_trigger_shipping

Parameter description: Specifies whether the trigger can be pushed to DNs for execution.

Type: USERSET

Value range: Boolean

- **on** indicates that the trigger can be pushed to DNs for execution.
- **off** indicates that the trigger cannot be pushed to DNs. It must be executed on the CN.

Default value: **on**

enable_remotejoin

Parameter description: Specifies whether JOIN operation plans can be delivered to DNs for execution.

Type: USERSET

Value range: Boolean

- **on** indicates that JOIN operation plans can be delivered to DNs for execution.
- **off** indicates that JOIN operation plans cannot be delivered to DNs for execution.

Default value: **on**

enable_remotegroup

Parameter description: Specifies whether the execution plans of **GROUP BY** and **AGGREGATE** can be delivered to DNs for execution.

Type: USERSET

Value range: Boolean

- **on** indicates that the execution plans of **GROUP BY** and **AGGREGATE** can be delivered to DNs for execution.

- **off** indicates that the execution plans of **GROUP BY** and **AGGREGATE** cannot be delivered to DNs for execution.

Default value: **on**

enable_remotelimit

Parameter description: Specifies whether the execution plan specified in the LIMIT clause can be pushed down to DNs for execution.

Type: USERSET

Value range: Boolean

- **on** indicates that the execution plan specified in the LIMIT clause can be pushed down to DNs for execution.
- **off** indicates that the execution plan specified in the LIMIT clause cannot be delivered to DNs for execution.

Default value: **on**

enable_limit_stop

Parameter description: Specifies whether the **early stop** optimization is enabled for **LIMIT** statements. For a **LIMIT n** statement, if **early stop** is used, the CN requests the DN to end the execution after receiving n pieces of data. This method is applicable to complex queries with **LIMIT**. This parameter is supported only by 8.1.3.320 and later cluster versions.

Type: USERSET

Value range: Boolean

- **on** indicates that **early stop** is enabled for LIMIT statements.
- **off** indicates that **early stop** is disabled for LIMIT statements.

Default value: **on**

enable_remotesort

Parameter description: Specifies whether the execution plan of the ORDER BY clause can be delivered to DNs for execution.

Type: USERSET

Value range: Boolean

- **on** indicates that the execution plan of the ORDER BY clause can be delivered to DNs for execution.
- **off** indicates that the execution plan of the ORDER BY clause cannot be delivered to DNs for execution.

Default value: **on**

enable_join_pseudoconst

Parameter description: Specifies whether joining with the pseudo constant is allowed. A pseudo constant indicates that the variables on both sides of a join are identical to the same constant.

Type: USERSET

Value range: Boolean

- **on** indicates that joining with the pseudo constant is allowed.
- **off** indicates that joining with the pseudo constant is not allowed.

Default value: **off**

cost_model_version

Parameter description: Specifies the model used for cost estimation in the application scenario. This parameter affects the distinct estimation of the expression, HashJoin cost model, estimation of the number of rows, distribution key selection during redistribution, and estimation of the number of aggregate rows.

Type: USERSET

Value range: **0**, **1**, or **2**

- **0** indicates that the original cost estimation model is used.
- **1** indicates that the enhanced distinct estimation of the expression, HashJoin cost estimation model, estimation of the number of rows, distribution key selection during redistribution, and estimation of the number of aggregate rows are used on the basis of **0**.
- **2** indicates that the ANALYZE sampling algorithm with better randomicity is used on the basis of **1** to improve the accuracy of statistics collection.

Default value: **1**

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(DWS) compilation.

Type: USERSET

Value range: Boolean

- **on** indicates that various assertion checks are enabled.
- **off** indicates that various assertion checks are disabled.

NOTE

This parameter is set to **on** by default if GaussDB(DWS) is compiled with various assertion checks enabled.

Default value: **off**

distribute_test_param

Parameter description: Specifies whether the embedded test stubs for testing the distribution framework take effect. In most cases, developers embed some test stubs in the code during fault injection tests. Each test stub is identified by a unique name. The value of this parameter is a triplet that includes three values: thread level, test stub name, and error level of the injected fault. The three values are separated by commas (,).

Type: USERSET

Value range: a string indicating the name of any embedded test stub.

Default value: **-1, default, default**

enable_crc_check

Parameter description: Specifies whether to enable data checks. Check information is generated when table data is written and is checked when the data is read. You are not advised to modify the settings.

Type: POSTMASTER

Value range: Boolean

- **on** indicates that data checks are enabled.
- **off** indicates that data checks are disabled.

Default value: **on**

NOTICE

If CRC is enabled, all data on a page must be written to WALs when hint bits of tuples on the page are modified for the first time after a checkpoint. This deteriorates the performance of the first query after the checkpoint.

ignore_checksum_failure

Parameter description: Sets whether to ignore check failures (but still generates an alarm) and continues reading data. This parameter is valid only when **enable_crc_check** is set to **on**. 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.

Type: SUSET

Value range: Boolean

- **on** indicates that data check errors are ignored.
- **off** indicates that data check errors are reported.

Default value: **off**

enable_colstore

Parameter description: Specifies whether to create a table as a column-store table by default when no storage method is specified. The value for each node must be the same. This parameter is used for tests. Users are not allowed to enable it.

Type: SUSET

Value range: Boolean

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. When **enable_force_vector_engine** is enabled, no matter it is a row-store table, column-store table, or hybrid row-column store table, if the plantree does not contain scenarios that do not support vectorization, the vectorized executor is forcibly used.

Type: USERSET

Value range: Boolean

Default value: **off**

enable_csqual_pushdown

Parameter description: Specifies whether to deliver filter criteria for a rough check during query.

Type: USERSET

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**

explain_dna_file

Parameter description: Specifies the name of a CSV file exported when **explain_perf_mode** is set to **run**.

Type: USERSET

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: **NULL**

explain_perf_mode

Parameter description: Specifies the display format of the **explain** command.

Type: USERSET

Value range: **normal**, **pretty**, **summary**, and **run**

- **normal** indicates that the default printing format is used.
- **pretty** indicates that the optimized display mode of GaussDB(DWS) is used. A new format contains a plan node ID, directly and effectively analyzing performance.
- **summary** indicates that the analysis result based on such information is printed in addition to the printed information in the format specified by **pretty**.
- **run** indicates that in addition to the printed information specified by **summary**, the database exports the information as a CSV file.

Default value: **pretty**

join_num_distinct

Parameter description: Controls the default distinct value of the join column or expression in application scenarios.

Type: USERSET

Value range: a double-precision floating point number greater than or equal to **-100**. Decimals may be truncated when displayed on clients.

- If the value is greater than **0**, the value is used as the default distinct value.
- If the value is greater than or equal to **-100** and less than **0**, it means the percentage used to estimate the default distinct value.
- If the value is **0**, the default distinct value is **200**.

Default value: **-20**

qual_num_distinct

Parameter description: Controls the default distinct value of the filter column or expression in application scenarios.

Type: USERSET

Value range: a double-precision floating point number greater than or equal to **-100**. Decimals may be truncated when displayed on clients.

- If the value is greater than **0**, the value is used as the default distinct value.
- If the value is greater than or equal to **-100** and less than **0**, it means the percentage used to estimate the default distinct value.
- If the value is **0**, the default distinct value is **200**.

Default value: **200**

trace_notify

Parameter description: Specifies whether to generate a large amount of debugging output for the **LISTEN** and **NOTIFY** commands. **client_min_messages** or **log_min_messages** must be **DEBUG1** or lower so that such output can be recorded in the logs on the client or server separately.

Type: USERSET

Value range: Boolean

- **on** indicates that the function is enabled.
- **off** indicates that the function is disabled.

Default value: **off**

trace_sort

Parameter description: Specifies whether to display information about resource usage during sorting operations in logs. This parameter is available only when the macro TRACE_SORT is defined during the GaussDB(DWS) compilation. However, TRACE_SORT is currently defined by default.

Type: USERSET

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(DWS) to report an error, aborting the current transaction.

Type: SUSET

Value range: Boolean

- **on** indicates that the function is enabled.
- **off** indicates that the function is disabled.

NOTE

- Setting this parameter to **on** causes the system to report a warning, pad the damaged page with zeros, and then continue with subsequent processing. This behavior will damage data, that is, all rows on the damaged page. However, it allows you to bypass the error and retrieve rows from any undamaged pages that are present in the table. Therefore, it is useful for restoring data that is damaged due to a hardware or software error. In most cases, you are not advised to set this parameter to **on** unless you do not want to restore data from the damaged pages of a table.
- For a column-store table, the system will skip the entire CU and then continue processing. The supported scenarios include the CRC check failure, magic check failure, and incorrect CU length.

Default value: **off**

replication_test

Parameter description: Specifies whether to enable internal testing on the data replication function.

Type: USERSET

Value range: Boolean

- **on** indicates that internal testing on the data replication function is enabled.
- **off** indicates that internal testing on the data replication function is disabled.

Default value: **off**

cost_param

Parameter description: Controls 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 (&) operations on the bit for each method. A method is selected if its value is not **0**.

If **cost_param & 1** is not set to **0**, an improvement mechanism is selected for calculating a non-equi join selection rate, which is more accurate in estimation of self-join (join between two same tables). In V300R002C00 and later, **cost_param & 1=0** is not used. That is, an optimized 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.

When **cost_param & 4** is not **0**, the selected debugging model is not recommended when the stream node is evaluated.

When **cost_param & 16** is not **0**, the model between fully correlated and fully uncorrelated models is used to calculate the comprehensive selection rate of two or more filtering conditions or join conditions. If there are many filtering conditions, the strongly-correlated model is preferred.

Type: USERSET

Value range: an integer ranging from 1 to INT_MAX

Default value: **16**

convert_string_to_digit

Parameter description: Specifies the implicit conversion priority, which determines whether to preferentially convert strings into numbers.

Type: USERSET

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

Modify this parameter only when absolutely necessary because the modification will change the rule for converting internal data types and may cause unexpected results.

nls_timestamp_format

Parameter description: Specifies the default timestamp format.

Type: USERSET

Value range: a string

Default value: **DD-Mon-YYYY HH:MI:SS.FF AM**

enable_partitionwise

Parameter description: Specifies whether to select an intelligent algorithm for joining partitioned tables.

Type: USERSET

Value range: Boolean

- **on** indicates that an intelligent algorithm is selected.
- **off** indicates that an intelligent algorithm is not selected.

Default value: **off**

enable_partition_dynamic_pruning

Parameter description: Specifies whether dynamic pruning is enabled during partition table scanning.

Type: USERSET

Value range: Boolean

- **on**: enable
- **off**: disable

Default value: **on**

max_user_defined_exception

Parameter description: Specifies the maximum number of exceptions. The default value cannot be changed.

Type: USERSET

Value range: an integer

Default value: **1000**

datanode_strong_sync

Parameter description: This parameter no longer takes effect.

Type: USERSET

Value range: Boolean

- **on** indicates that forcible synchronization between stream nodes is enabled.
- **off** indicates that forcible synchronization between stream nodes is disabled.

Default value: **off**

enable_global_stats

Parameter description: Specifies the current statistics mode. This parameter is used to compare global statistics generation plans and the statistics generation plans for a single DN. This parameter is used for tests. Users are not allowed to enable it.

Type: SUSET

Value range: Boolean

- **on** or **true** indicates the global statistics mode.
- **off** or **false** indicates the single-DN statistics mode.

Default value: **on**

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.

Type: USERSET

Value range: Boolean

- **on/true** indicates that optimization for numeric data calculation is enabled.
- **off/false** indicates that optimization for numeric data calculation is disabled.

Default value: **on**

enable_row_fast_numeric

Parameter description: Specifies the format in which numeric data in a row-store table is spilled to disks.

Type: USERSET

Value range: Boolean

- **on/true** indicates that numeric data in a row-store table is spilled to disks in bigint format.
- **off/false** indicates that numeric data in a row-store table is spilled to disks in the original format.

NOTICE

If this parameter is set to **on**, you are advised to enable **enable_force_vector_engine** to improve the query performance of large data sets. However, compared with the original format, there is a high probability that the bigint format occupies more disk space. For example, the TPC-H test set occupies about 7% more space (reference value, may vary depending on the environment).

Default value: **off**

rewrite_rule

Parameter description: Specifies the rewriting rule for enabled optional queries. Some query rewriting rules are optional. Enabling them cannot always improve query efficiency. In a specific customer scenario, you can set the query rewriting rules through the GUC parameter to achieve optimal query efficiency.

This parameter can control the combination of query rewriting rules, for example, there are multiple rewriting rules: rule1, rule2, rule3, and rule4. To set the parameters, you can perform the following operations:

```
set rewrite_rule=rule1;          --Enable query rewriting rule rule1.
set rewrite_rule=rule2,rule3;    --Enable query rewriting rules rule2 and rule3.
set rewrite_rule=none;           --Disable all optional query rewriting rules.
```

Type: USERSET

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 (from the main query to subqueries).
- **uniquecheck**: Uses the Unique Check rewriting rule. (The scenario where the target column does not contain the expression sublink of the aggregate function can be improved. The function can be enabled only when the value of the target column is unique after the sublink is aggregated based on the associated column. This function is recommended to be used by optimization engineers.)
- **disablerep**: Uses the function that prohibits pulling up sublinks of the replication table. (Disables sublink pull-up for the replication table.)
- **projection_pushdown**: the Projection Pushdown rewriting rule (Removes columns that are not used by the parent query from the subquery).
- **or_conversion**: the OR conversion rewriting rule (eliminates the association OR conditions that are inefficient to execute).

- **plain_lazyagg**: Uses the **Plain Lazy Agg** query rewriting rule (eliminates aggregation operations in a single subquery). This option is supported only by clusters of version 8.1.3.100 or later.

Default value: **magicset**, **or_conversion**, **projection_pushdown**, and **plain_lazyagg**

enable_compress_spill

Parameter description: Specifies whether to enable the compression function of writing data to a disk.

Type: USERSET

Value range: Boolean

- **on/true** indicates that optimization for writing data to a disk is enabled.
- **off/false** indicates that optimization for writing data to a disk is disabled.

Default value: **on**

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.

Type: USERSET

Value range: a string

- **LLVM_COMPILE** indicates that the codegen compilation time of each thread is displayed on the explain performance page.
- **HASH_CONFLICT** indicates that the log file in the **pg_log** directory of the DN process displays the hash table statistics, including the hash table size, hash chain length, and hash conflict information.
- **STREAM_DATA_CHECK** indicates that a CRC check is performed on data before and after network data transmission.

Default value: **off(ALL)**, which indicates that no location function is enabled.

resource_track_log

Parameter description: Specifies the log level of self-diagnosis. Currently, this parameter takes effect only in multi-column statistics.

Type: USERSET

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**

hll_default_log2m

Parameter description: Specifies the number of buckets for HLL data. Using more buckets in HLL calculations leads to more precise and less deviated distinct value results. The deviation range is as follows: [–1.04/$2^{log2m*1/2}$, +1.04/$2^{log2m*1/2}$]

Type: USERSET

Value range: an integer ranging from 10 to 16

Default value: **11**

hll_default_regwidth

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. For details, see **Table 15-4**.

Type: USERSET

Value range: an integer ranging from 1 to 5

Default value: **5**

Table 15-4 Maximum number of calculated distinct values determined by hll_default_log2m and hll_default_regwidth

log2m	regwidth = 1	regwidth = 2	regwidth = 3	regwidth = 4	regwidth = 5
10	7.4e+02	3.0e+03	4.7e+04	1.2e+07	7.9e+11
11	1.5e+03	5.9e+03	9.5e+04	2.4e+07	1.6e+12
12	3.0e+03	1.2e+04	1.9e+05	4.8e+07	3.2e+12
13	5.9e+03	2.4e+04	3.8e+05	9.7e+07	6.3e+12
14	1.2e+04	4.7e+04	7.6e+05	1.9e+08	1.3e+13
15	2.4e+04	9.5e+04	1.5e+06	3.9e+08	2.5e+13

hll_default_expthresh

Parameter description: Specifies the default threshold for switching from the **explicit** mode to the **sparse** mode.

Type: USERSET

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

Parameter description: Specifies whether to enable the **sparse** mode by default.

Type: USERSET

Valid value: **0** and **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

Parameter description: Specifies the size of **max_sparse**.

Type: USERSET

Value range: an integer ranging from –1 to **INT_MAX**

Default value: **–1**

enable_compress_hll

Parameter description: Specifies whether to enable memory optimization for HLL.

Type: USERSET

Value range: Boolean

- **on** or **true** indicates that memory optimization is enabled.
- **off** or **false** indicates that memory optimization is disabled.

Default value: **off**

udf_memory_limit

Parameter description: Controls the maximum physical memory that can be used when each CN or DN executes UDFs.

Type: POSTMASTER

Value range: an integer ranging from 200 x 1024 to the value of **max_process_memory** and the unit is KB.

Default value: **0.05 * max_process_memory**

FencedUDFMemoryLimit

Parameter description: Controls the virtual memory used by each fenced udf worker process.

Type: USERSET

Suggestion: You are not advised to set this parameter. You can set **udf_memory_limit** instead.

Value range: an integer. The unit can be KB, MB, or GB. **0** indicates that the memory is not limited.

Default value: **0**

UDFWorkerMemHardLimit

Parameter description: Specifies the maximum value of **fencedUDFMemoryLimit**.

Type: POSTMASTER

Suggestion: You are not advised to set this parameter. You can set **udf_memory_limit** instead.

Value range: an integer. The unit can be KB, MB, or GB.

Default value: **1 GB**

enable_pbe_optimization

Parameter description: Specifies whether the optimizer optimizes the query plan for statements executed in Parse Bind Execute (PBE) mode.

Type: USERSET

Value range: Boolean

- **on** indicates that the optimizer optimizes the query plan.
- **off** indicates that the optimization does not optimize the query plan.

Default value: **on**

enable_light_proxy

Parameter description: Specifies whether the optimizer optimizes the execution of simple queries on CNs.

Type: USERSET

Value range: Boolean

- **on** indicates that the optimizer optimizes the execution.
- **off** indicates that the optimization does not optimize the execution.

Default value: **on**

checkpoint_flush_after

Parameter description: Specifies the number of consecutive disk pages that the checkpointer writer thread writes before asynchronous flush. In GaussDB(DWS), the size of a disk page is 8 KB.

Type: SIGHUP

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: **32**

enable_parallel_ddl

Parameter description: Controls whether multiple CNs can concurrently perform DDL operations on the same database object.

Type: USERSET

Value range: Boolean

- **on** indicates that DDL operations can be performed safely and that no distributed deadlock occurs.
- **off** indicates that DDL operations cannot be performed safely and that distributed deadlocks may occur.

Default value: **on**

gc_fdw_verify_option

Parameter description: Specifies whether to enable the logic for verifying the number of rows in a result set in the collaborative analysis. This parameter is supported by version 8.1.3.310 or later clusters.

Type: USERSET

Value range: Boolean

- **on** indicates that the logic for verifying the number of rows in the result set is enabled. The **SELECT COUNT** statement is used to obtain the expected number of rows and compare it with the actual number of rows.
- **off** indicates that the logic for verifying the number of rows in the result set is disabled and only the required result set is obtained.

Default value: **on**

NOTE

- If this parameter is enabled, the performance deteriorates slightly. In performance-sensitive scenarios, you can disable this parameter to improve the performance.
- If an exception is thrown during the result set row verification. You can set **log_min_messages=debug1** and **logging_module='on(COOP_ANALYZE)'** to obtain the collaborative analysis logs.

show_acce_estimate_detail

Parameter description: When the GaussDB(DWS) cluster is accelerated (**acceleration_with_compute_pool** is set to **on**), specifies whether the **EXPLAIN** statement displays the evaluation information about execution plan pushdown to computing Node Groups. 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.

Type: USERSET

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**

support_batch_bind

Parameter description: Specifies whether to batch bind and execute PBE statements through interfaces such as JDBC, ODBC, and Libpq.

Type: SIGHUP

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**

15.22 Auditing

15.22.1 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.

Type: SIGHUP

Value range: Boolean

- **on** indicates that the auditing function is enabled.
- **off** indicates that the auditing function is disabled.

Default value: **on**

audit_space_limit

Parameter description: Specifies the total disk space occupied by audit files.

Type: SIGHUP

Value range: an integer ranging from **1024 KB** to **1024 GB**. The unit is KB.

Default value: **1GB**

15.22.2 Operation Audit

audit_system_object

Parameter description: Specifies whether to audit the CREATE, DROP, and ALTER operations on the GaussDB(DWS) database object. The GaussDB(DWS) database objects include databases, users, schemas, and tables. The operations on the database object can be audited by changing the value of this parameter.

Type: SIGHUP

Value range: an integer ranging from 0 to 4194303

- **0** indicates that the function of auditing the CREATE, DROP, and ALTER operations on the GaussDB(DWS) database object can be disabled.
- Other values indicate that the CREATE, DROP, and ALTER operations on a certain or some GaussDB(DWS) database objects are audited.

Value description:

The value of this parameter is calculated by 22 binary bits. The 22 binary bits represent 22 types of GaussDB(DWS) database 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 content represented by these 22 binary bits, see **Table 15-5**.

Default value: **12303**

Table 15-5 Meaning of each value for the **audit_system_object** parameter

Binary Bit	Meaning	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.

Binary Bit	Meaning	Value Description
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, DROP, and ALTER operations on views.	● **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 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.

Binary Bit	Meaning	Value Description
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	Whether to audit the CREATE, DROP, and ALTER operations on SERVER FOR HADOOP 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 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	Whether to audit the CREATE, DROP, and ALTER operations on Node Groups.	• **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 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 (configurations and dictionaries)	• **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.

Binary Bit	Meaning	Value Description
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 workloads.	● **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 redaction policies.	● **0** indicates that the CREATE, DROP, and ALTER operations on redaction policies are not audited. ● **1** indicates that the CREATE, DROP, and ALTER operations on redaction policies are audited.
Bit 20	Whether to audit the CREATE, DROP, and ALTER operations on sequences.	● **0** indicates that the CREATE, DROP, and ALTER operations on sequences are not audited. ● **1** indicates that the CREATE, DROP, and ALTER operations on sequences are audited.
Bit 21	Whether to audit the CREATE, DROP, and ALTER operations on nodes.	● **0** indicates that the CREATE, DROP, and ALTER operations on nodes are not audited. ● **1** indicates that the CREATE, DROP, and ALTER operations on nodes are audited.

enableSeparationOfDuty

Parameter description: Specifies whether the separation of permissions is enabled.

Type: POSTMASTER

Value range: Boolean

- **on** indicates that the separation of permissions is enabled.
- **off** indicates that the separation of permissions is disabled.

Default value: **off**

enable_grant_option

Parameter description: Specifies whether the **with grant option** function can be used in security mode.

Type: SIGHUP

Value range: Boolean

- **on** indicates that the **with grant option** function can be used in security mode.
- **off** indicates that the **with grant option** function cannot be used in security mode.

Default value: **off**

enable_grant_public

Parameter description: Specifies whether to allow the **grant to public** function in security mode.

Type: SIGHUP

Value range: Boolean

- **on** indicates that the **grant to public** function can be used in security mode.
- **off** indicates that the **grant to public** function cannot be used in security mode.

Default value: **off**

15.23 Transaction Monitoring

The automatic rollback transaction can be monitored and its statement problems can be located by setting the transaction timeout warning. In addition, the statements with long execution time can also be monitored.

transaction_sync_naptime

Parameter description: For data consistency, when the local transaction's status differs from that in the snapshot of the GTM, other transactions will be blocked. You need to wait for a few minutes until the transaction status of the local host is consistent with that of the GTM. The **gs_clean** tool is automatically triggered for cleansing when the waiting period on the CN exceeds that of **transaction_sync_naptime**. The tool will shorten the blocking time after it completes the cleansing.

Type: USERSET

Value range: an integer. The minimum value is **0**. The unit is second.

Default value: **5s**

NOTE

If the value of this parameter is set to **0**, gs_clean will not be automatically invoked for the cleansing before the blocking arrives the duration. Instead, the gs_clean tool is invoked by gs_clean_timeout. The default value is 5 minutes.

transaction_sync_timeout

Parameter description: For data consistency, when the local transaction's status differs from that in the snapshot of the GTM, other transactions will be blocked. You need to wait for a few minutes until the transaction status of the local host is consistent with that of the GTM. An exception is reported when the waiting duration on the CN exceeds the value of **transaction_sync_timeout**. Roll back the transaction to avoid system blocking due to long time of process response failures (for example, sync lock).

Type: USERSET

Value range: an integer. The minimum value is **0**. The unit is second.

Default value: **10min**

NOTE

- If the value is **0**, no error is reported when the blocking times out or the transaction is rolled back.
- The value of this parameter must be greater than **gs_clean_timeout**. Otherwise, unnecessary transaction rollback will probably occur due to a block timeout caused by residual transactions that have not been deleted by **gs_clean** on a DN.

15.24 GTM Parameters

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.

NOTICE

If the values of **client_min_messages** and **log_min_messages** are the same, they indicate different levels.

Type: SUSET

Valid values: enumerated values. Valid values are **debug**, **debug5**, **debug4**, **debug3**, **debug2**, **debug1**, **info**, **log**, **notice**, **warning**, **error**, **fatal**, and **panic**. For details about the parameters, see **Table 15-3**.

Default value: **warning**

enable_alarm

Parameter description: Specifies whether to enable the alarm detection thread to detect the fault scenarios that may occur in the database.

Type: POSTMASTER

Value range: Boolean

- **on**: Alarm detection thread is enabled.
- **off**: Alarm detection thread is disabled.

Default value: **on**

15.25 Miscellaneous Parameters

enable_cluster_resize

Parameter description: Indicates whether the current session is a scale-out redistribution session. This parameter applies only to scale-out redistribution sessions. Do not set this parameter for other service sessions.

Parameter type: SUSET

Value range: Boolean

- **on** indicates that the current session is for scaling or redistributing data, and allows the execution of specific SQL statements for redistribution.
- **off** indicates that the current session is not for scaling or redistributing data, and does not allow the execution of specific SQL statements for redistribution.

Default value: **off**

NOTE

This parameter is used for internal O&M. Do not set it to **on** unless absolutely necessary.

dfs_partition_directory_length

Parameter description: Specifies the largest directory name length for the partition directory of a table partitioned by VALUE in the HDFS.

Type: USERSET

Value range: 92 to 7999

Default value: **512**

enable_hadoop_env

Parameter description: Sets whether local row- and column-store tables can be created in a database while the Hadoop feature is used. In the GaussDB(DWS) cluster, it is set to **off** by default to support local row- and column- based storage and cross-cluster access to Hadoop. You are not advised to change the value of this parameter.

Type: USERSET

Value range: Boolean

- **on** or **true**, indicating that local row- and column-store tables cannot be created in a database while the Hadoop feature is used.
- **off** or **false**, indicating that local row- and column-based tables can be created in a database while the Hadoop feature is used.

Default value: **off**

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 avoided when update and delete operations are performed at the same time.

Type: USERSET

Value range: Boolean

- If this parameter is set to **on**, the delta merge operation internally increases the lock level. In this way, when any two of the **DELTAMERGE**, **UPDATE**, and **DELETE** operations are concurrently performed, an operation can be performed only after the previous one is complete.
- If this parameter is set to **off**, and any two of the **DELTAMERGE**, **UPDATE**, and **DELETE** operations are concurrently performed to data in a row in the delta table of the HDFS table, errors will be reported during the later operation, and the operation will stop.

Default value: **off**

job_queue_processes

Parameter description: Specifies the number of jobs that can be concurrently executed.

Type: POSTMASTER

Value range: 0 to 1000

Functions:

- Setting **job_queue_processes** to **0** indicates that the scheduled task function is disabled and that no job will be executed. (Enabling scheduled tasks 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 task function is enabled and this value is the maximum number of tasks that can be concurrently processed.

After the scheduled task function is enabled, the **job_scheduler** thread at a scheduled interval polls the **pg_jobs** system catalog. The scheduled task check is performed every second by default.

Too many concurrent tasks consume many system resources, so you need to set the number of concurrent tasks to be processed. If the current number of concurrent tasks reaches **job_queue_processes** and some of them expire, these tasks 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 task to avoid the problem that tasks in the next polling period cannot be properly processed because overlong task execution time.

Note: If the number of parallel 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**

ngram_gram_size

Parameter description: Specifies the length of the ngram parser segmentation.

Type: USERSET

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.

Type: USERSET

Value range: Boolean

- **on**: Ignores graphical characters.
- **off**: Does not ignore graphical characters.

Default value: **off**

ngram_punctuation_ignore

Parameter description: Specifies whether the ngram parser ignores punctuations.

Type: USERSET

Value range: Boolean

- **on**: Ignores punctuations.
- **off**: Does not ignore punctuations.

Default value: **on**

zhparser_dict_in_memory

Parameter description: Specifies whether Zhparser adds a dictionary to memory.

Type: POSTMASTER

Value range: Boolean

- **on**: Adds the dictionary to memory.
- **off**: Does not add the dictionary to memory.

Default value: **on**

zhparser_multi_duality

Parameter description: Specifies whether Zhparser aggregates segments in long words with duality.

Type: USERSET

Value range: Boolean

- **on**: Aggregates segments in long words with duality.
- **off**: Does not aggregate segments in long words with duality.

Default value: **off**

zhparser_multi_short

Parameter description: Specifies whether Zhparser executes long words composite divide.

Type: USERSET

Value range: Boolean

- **on**: Performs compound segmentation for long words.
- **off**: Does not perform compound segmentation for long words.

Default value: **on**

zhparser_multi_zall

Parameter description: Specifies whether Zhparser displays all single words individually.

Type: USERSET

Value range: Boolean

- **on**: Displays all single words separately.
- **off**: Does not display all single words separately.

Default value: **off**

zhparser_multi_zmain

Parameter description: Specifies whether Zhparser displays important single words separately.

Type: USERSET

Value range: Boolean

- **on**: Displays important single words separately.

- **off**: Does not display important single words separately.

Default value: **off**

zhparser_punctuation_ignore

Parameter description: Specifies whether the Zhparser segmentation result ignores special characters including punctuations (\r and \n will not be ignored).

Type: USERSET

Value range: Boolean

- **on**: Ignores all the special characters including punctuations.
- **off**: Does not ignore all the special characters including punctuations.

Default value: **on**

zhparser_seg_with_duality

Parameter description: Specifies whether Zhparser aggregates segments in long words with duality.

Type: USERSET

Value range: Boolean

- **on**: Aggregates segments in long words with duality.
- **off**: Does not aggregate segments in long words with duality.

Default value: **off**

acceleration_with_compute_pool

Parameter description: Specifies whether to use the computing resource pool for acceleration when OBS is queried.

Type: USERSET

Value range: Boolean

- **on** indicates that the query covering 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**

behavior_compat_options

Parameter description: Specifies database compatibility behavior. Multiple items are separated by commas (,).

Type: USERSET

Value range: a string

Default value: In upgrade scenarios, the default value of this parameter is the same as that in the cluster before the upgrade. When a new cluster is installed, the default value of this parameter is **check_function_conflicts** to prevent serious problems caused by incorrect function attributes defined by users.

NOTE

- Currently, only **Table 15-6** is supported.
- Multiple items are separated by commas (,), for example, **set behavior_compat_options='end_month_calculate,display_leading_zero';**
- **strict_concat_functions** and **strict_text_concat_td** are mutually exclusive.

Table 15-6 Compatibility configuration items

Configuration Item	Behavior	Applicable Compatibility Mode

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**.
	For example, during data migration, if this parameter is not set during data import, when floating numbers are displayed or converted to strings, the leading zeros of the floating point numbers are omitted, causing an error message like this:
	`ERROR: xxx invalid input syntax for type xxx`
	`DETAIL: Token "." is invalid`
		ORA
		TD

Configuration Item	Behavior	Applicable Compatibility Mode

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 sum 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:
	`SELECT add_months('2018-02-28',3) FROM dual;`
	`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:
	`SELECT add_months('2018-02-28',3) FROM dual;`
	`add_months`
	`---------------------`
	`2018-05-31 00:00:00`
	`(1 row)`
		ORA
		TD

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 CN memory consumption and maintaining the stability of ANALYZE.	ORA TD MySQL
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*.	ORA TD MySQL

Configuration Item	Behavior	Applicable Compatibility Mode
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.	ORA TD MySQL
correct_to_number	Controls 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 Oracle.	ORA

unbind_divide_bound	Controls the range check on the result of integer division.
	● If this item is not specified, the division result is checked. If the result is out of the range, an error is reported. In the following example, an out-of-range error is reported because the value of **INT_MIN/(-1)** is greater than the value of **INT_MAX**.
	`SELECT (-2147483648)::int / (-1)::int;`
	`ERROR:  integer out of range`
	● If this item is specified, the range of the division result does not need to be checked. In the following example, **INT_MIN/(-1)** can be used to obtain the output result **INT_MAX+1**.
	`SELECT (-2147483648)::int / (-1)::int;`
	`  ?column?`
	`------------`
	` 2147483648`
	`(1 row)`
		ORA
		TD

merge_update_multi	Specifies whether to perform an update when **MERGE INTO** is executed to match multiple rows. If this item is specified, no error is reported when multiple rows are matched. Otherwise, an error is reported (same as Oracle).	ORA TD

Configuration Item	Behavior	Applicable Compatibility Mode
disable_row_update_multi	Specifies whether to perform an update when multiple rows of a row-store table are matched. If this item is specified, an error is reported when multiple rows are matched. Otherwise, multiple rows can be matched and updated by default.	ORA TD

return_null_string	Specifies how to display the empty result (empty string '') of the lpad(), rpad(), repeat(), regexp_split_to_table(), and split_part() functions.
	● If this item is not specified, the empty string is displayed as **NULL**.
	`SELECT length(lpad('123',0,'*')) FROM dual;`
	`length`
	`--------`
	
	`(1 row)`
	● If this item is specified, the empty string is displayed as single quotation marks ('').
	`SELECT length(lpad('123',0,'*')) FROM dual;`
	`length`
	`--------`
	`0`
	`(1 row)`
		ORA

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 Oracle and Teradata are retained. If this item is not specified and a **concat** function has a parameter of the **variadic** type, the result format of Oracle is retained for both Oracle and Teradata.	ORA TD

Configuration Item	Behavior	Applicable Compatibility Mode
convert_string_digit_to_numeric	Specifies the type casting priority for binary BOOL operations on the CHAR type and INT type. ● If this item is not specified, the type casting priority is the same as that of PG9.6. ● After this item is configured, all binary BOOL operations of the CHAR type and INT type are forcibly converted to the NUMERIC type for computation. After this configuration item is set, the CHAR types that are affected include BPCHAR, VARCHAR, NVARCHAR2, and TEXT, and the INT types that are affected include INT1, INT2, INT4, and INT8. **CAUTION** This configuration item is valid only for binary BOOL operation, for example, **INT2>TEXT** and **INT4=BPCHAR**. Non-BOOL operation is not affected. This configuration item does not support conversion of UNKNOWN operations such as **INT>'1.1'**. After this configuration item is enabled, all BOOL operations of the CHAR and INT types are preferentially converted to the NUMERIC type for computation, which affects the computation performance of the database. When the JOIN column is a combination of affected types, the execution plan is affected.	ORA TD MySQL

Configuration Item	Behavior	Applicable Compatibility Mode

check_function_conflicts	Controls the check of the custom **plpgsql/SQL** function attributes.
	● If this parameter is not specified, the **IMMUTABLE/STABLE/VOLATILE** attributes of a custom function are not checked.
	● If this parameter is specified, the IMMUTABLE attribute of a custom function is checked. If the function contains a table or the STABLE/ VOLATILE function, an error is reported during the function execution. In a custom function, a table or the STABLE/VOLATILE function conflicts with the IMMUTABLE attribute, thus function behaviors are not IMMUTABLE in this case.
	For example, when this parameter is specified, an error is reported in the following scenarios:
	`CREATE OR replace FUNCTION sql_immutable (INTEGER)`
	`RETURNS INTEGER AS 'SELECT a+$1 FROM shipping_schema.t4 WHERE a=1;'`
	`LANGUAGE SQL IMMUTABLE`
	`RETURNS NULL`
	`ON NULL INPUT;`
	`select sql_immutable(1);`
	`ERROR:  IMMUTABLE function cannot contain SQL statements with relation or Non-IMMUTABLE function.`
	`CONTEXT:  SQL function "sql_immutable" during startup referenced column: sql_immutable`
		ORA
		TD
		MySQL

Configuration Item	Behavior	Applicable Compatibility Mode

varray_verification	Indicates whether to verify the array length and array type length. Compatible with GaussDB(DWS) versions earlier than 8.1.0.
	If this parameter is specified, the array length and array type length are not verified.
	`Scenario 1`
	`CREATE OR REPLACE PROCEDURE varray_verification`
	`AS`
	`    TYPE org_varray_type IS varray(5) OF VARCHAR2(2);`
	`    v_org_varray org_varray_type;`
	`BEGIN`
	`    v_org_varray(1) := '111'; --If the value exceeds the limit of` **VARCHAR2(2)**`, the setting will be consistent with that in the historical version and no verification is performed after configuring this option.`
	`END;`
	`/`
	`Scenario 2`
	`CREATE OR REPLACE PROCEDURE varray_verification_i3_1`
	`AS`
	`    TYPE org_varray_type IS varray(2) OF NUMBER(2);`
	`    v_org_varray org_varray_type;`
	`BEGIN`
	`    v_org_varray(3) := 1; --If the value exceeds the limit of` **varray(2)** `specified for array length, the setting will be consistent with that in the historical version and no verification is performed after configuring this option.`
	`END;`
	`/`
		ORA
		TD

Configuration Item	Behavior	Applicable Compatibility Mode

strict_concat_functions	Indicates whether the **textanycat()** and **anytextcat()** functions are compatible with the return value if there are null parameters. This parameter and **strict_text_concat_td** are mutually exclusive.
	In MySQL-compatible mode, this parameter has no impact.
	● If this configuration item is not specified, the returned values of the **textanycat()** and **anytextcat()** functions are the same as those in the Oracle database.
	● When this configuration item is specified, if there are null parameters in the **textanycat()** and **anytextcat()** functions, the returned value is also null. Different result formats in Oracle and Teradata are retained.
	If this configuration item is not specified, the returned values of the **textanycat()** and **anytextcat()** functions are the same as those in the Oracle database.
	`SELECT textanycat('gauss', cast(NULL as BOOLEAN));`
	` textanycat`
	`------------`
	` gauss`
	`(1 row)`
	
	`SELECT 'gauss' || cast(NULL as BOOLEAN); -- In this case, the || operator is converted to the textanycat function.`
	` ?column?`
	`----------`
	` gauss`
	`(1 row)`
	When setting this configuration item, retain the results that are different from those in Oracle and Teradata:
	`SELECT textanycat('gauss', cast(NULL as BOOLEAN));`
	` textanycat`
	`------------`
	
	`(1 row)`
	
	`SELECT 'gauss' || cast(NULL as BOOLEAN); -- In this case, the || operator is converted to the textanycat function.`
	` ?column?`
	`----------`
	
	`(1 row)`
		ORA
		TD

Configuration Item	Behavior	Applicable Compatibility Mode

strict_text_concat_td	In Teradata compatible mode, whether the **textcat()**, **textanycat()** and **anytextcat()** functions are compatible with the return value if there are null parameters. This parameter and **strict_concat_functions** are mutually exclusive.
	● If this parameter is not specified, the return values of the **textcat()**, **textanycat()**, and **anytextcat()** functions in Teradata-compatible mode are the same as those in GaussDB(DWS).
	● When this parameter is specified, if the **textcat()**, **textanycat()**, and **anytextcat()** functions contain any null parameter values, the return value is null in the Teradata-compatible mode.
	If this parameter is not specified, the returned values of the **textcat()**, **textanycat()**, and **anytextcat()** functions are the same as those in the GaussDB(DWS).
	`td_compatibility_db=# SELECT textcat('abc', NULL);`
	`textcat`
	`---------`
	`abc`
	`(1 row)`
	`td_compatibility_db=# SELECT 'abc' || NULL; -- In this case, the operator` **||** `is converted to the` **textcat()** `function.`
	`?column?`
	`----------`
	`abc`
	`(1 row)`
	When this parameter is specified, **NULL** is returned if any of the **textcat()**, **textanycat()**, and **anytextcat()** functions returns a null value.
	`td_compatibility_db=# SELECT textcat('abc', NULL);`
	`textcat`
	`---------`
	
	`(1 row)`
	`td_compatibility_db=# SELECT 'abc' || NULL;`
	`?column?`
	`----------`
	
	`(1 row)`
		TD

Configuration Item	Behavior	Applicable Compatibility Mode

compat_display_ref_table	Sets the column display format in the view.
	● If this parameter is not specified, the prefix is used by default, in the **tab.col** format.
	● Specify this parameter to the same original definition. It is displayed only when the original definition contains a prefix.
	`SET behavior_compat_options='compat_display_ref_table';`
	`CREATE OR REPLACE VIEW viewtest2 AS SELECT a.c1, c2, a.c3, 0 AS c4 FROM viewtest_tbl a;`
	`SELECT pg_get_viewdef('viewtest2');`
	`pg_get_viewdef`
	`-----------------------------------------------------`
	`SELECT a.c1, c2, a.c3, 0 AS c4 FROM viewtest_tbl a;`
	`(1 row)`
		ORA
		TD

para_support_set_func	Whether the input parameters of the **COALESCE()**, **NVL()**, **GREATEST()**, and **LEAST()** functions in a column-store table support multiple result set expressions.
	● If this item is not specified and the input parameter contains multiple result set expressions, an error is reported, indicating that the function is not supported.
	`SELECT COALESCE(regexp_split_to_table(c3,'#'), regexp_split_to_table(c3,'#')) FROM regexp_ext2_tb1 ORDER BY 1 LIMIT 5;`
	`ERROR:  set-valued function called in context that cannot accept a set`
	● When this configuration item is specified, the function input parameter can contain multiple result set expressions.
	`SELECT COALESCE(regexp_split_to_table(c3,'#'), regexp_split_to_table(c3,'#')) FROM regexp_ext2_tb1 ORDER BY 1 LIMIT 5;`
	` coalesce`
	`----------`
	` a`
	` a`
	` a`
	` a`
	` a`
	`(5 rows)`
		ORA
		TD

Configuration Item	Behavior	Applicable Compatibility Mode
disable_select_truncate_parallel	Controls the DDL lock level such as TRUNCATE in a partitioned table. ● If this item is specified, the concurrent execution of TRUNCATE and DML operations (such as SELECT) on different partitions is forbidden, and the fast query shipping (FQS) of the SELECT operation on the partitioned table is allowed. You can set this parameter in the OLTP database, where there are many simple queries on partitioned tables, and there is no requirement for concurrent TRUNCATE and DML operations on different partitions. ● If this item is not specified, SELECT and TRUNCATE operations can be concurrently performed on different partitions in a partitioned table, and the FQS of the partitioned table is disabled to avoid possible inconsistency.	ORA TD MySQL

bpchar_text_without_rtrim	In Teradata-compatible mode, controls the space to be retained on the right during the character conversion from **bpchar** to **text**. If the actual length is less than the length specified by **bpchar**, spaces are added to the value to be compatible with the Teradata style of the **bpchar** character string.
	Currently, ignoring spaces at the end of a string for comparison is not supported. If the concatenated string contains spaces at the end, the comparison is space-sensitive.
	The following is an example:
	`td_compatibility_db=# SELECT length('a'::char(10)::text);`
	`length`
	`--------`
	`10`
	`(1 row)`
	
	`td_compatibility_db=# SELECT length('a'||'a'::char(10));`
	`length`
	`--------`
	`11`
	`(1 row)`
		TD

Configuration Item	Behavior	Applicable Compatibility Mode

convert_empty_str_to_null_td	In Teradata-compatible mode, controls the **to_date**, **to_timestamp**, and **to_number** type conversion functions to return **null** when they encounter empty strings, and controls the format of the return value when the **to_char** function encounters an input parameter of the date type.
	Example:
	If this parameter is not specified:
	`td_compatibility_db=# SELECT to_number('');`
	` to_number`
	`-----------`
	`         0`
	`(1 row)`
	
	`td_compatibility_db=# SELECT to_date('');`
	`ERROR:  the format is not correct`
	`DETAIL:  invalid date length "0", must between 8 and 10.`
	`CONTEXT:  referenced column: to_date`
	
	`td_compatibility_db=# SELECT to_timestamp('');`
	`      to_timestamp`
	`------------------------`
	` 0001-01-01 00:00:00 BC`
	`(1 row)`
	
	`td_compatibility_db=# SELECT to_char(date '2020-11-16');`
	`        to_char`
	`------------------------`
	` 2020-11-16 00:00:00+08`
	`(1 row)`
	If this parameter is specified, and parameters of **to_number**, **to_date**, and **to_timestamp** functions contain empty strings:
	`td_compatibility_db=# SELECT to_number('');`
	` to_number`
	`-----------`
	
	`(1 row)`
	
	`td_compatibility_db=# SELECT to_date('');`
	` to_date`
	`---------`
	
	`(1 row)`
	
	`td_compatibility_db=# SELECT to_timestamp('');`
	` to_timestamp`
	`--------------`
	
	`(1 row)`
	
	`td_compatibility_db=# SELECT to_char(date '2020-11-16');`
	`  to_char`
	`------------`
	` 2020/11/16`
	`(1 row)`
		TD

Configuration Item	Behavior	Applicable Compatibility Mode
disable_case_specific	Determines whether to ignore case sensitivity during character type match. This parameter is valid only in Teradata-compatible mode. ● If this item is not specified, characters are case sensitive during character type match. ● If this item is specified, characters are case insensitive during character type match. After being specified, this item will affect five character types (**CHAR**, **TEXT**, **BPCHAR**, **VARCHAR**, and **NVARCHAR**), 12 operators (**<**, **>**, **=**, **>=**, **<=**, **!=**, **<>**, **!=**, **like**, **not like**, **in**, and **not in**), and expressions **case when** and **decode**. **CAUTION** After this item is enabled, the **UPPER** function is added before the character type, which affects the estimation logic. Therefore, an enhanced estimation model is required. (Suggested settings: **cost_param=16**, **cost_model_version = 1**, **join_num_distinct=-20**, and **qual_num_distinct=200**)	TD

enable_interval_to_text	Controls the implicit conversion from the **interval** type to the **text** type.
	● When this option is enabled, the implicit conversion from the **interval** type to the **text** type is supported.
	`SELECT TO_DATE('20200923', 'yyyymmdd') - TO_DATE('20200920', 'yyyymmdd') = '3'::text;`
	`?column?`
	`----------`
	`f`
	`(1 row)`
	● When this option is disabled, the implicit conversion from the **interval** type to the **text** type is not supported.
	`SELECT TO_DATE('20200923', 'yyyymmdd') - TO_DATE('20200920', 'yyyymmdd') = '3'::text;`
	`?column?`
	`----------`
	`t`
	`(1 row)`
		ORA
		TD
		MySQL

Configuration Item	Behavior	Applicable Compatibility Mode

case_insensitive	In MySQL-compatible mode, configure this parameter to specify the case-insensitive input parameters of the **locate**, **strpos**, and **instr** string functions.
	Currently, this parameter is not configured by default. That is, the input parameter is case-sensitive.
	The following shows an example:
	● If this parameter is not configured, the input parameter is case-sensitive.
	`mysql_compatibility_db=# SELECT LOCATE('sub', 'Substr');`
	` locate`
	`--------`
	`      0`
	`(1 row)`
	● If this parameter is configured, the input parameter is case-insensitive.
	`mysql_compatibility_db=# SELECT LOCATE('sub', 'Substr');`
	` locate`
	`--------`
	`      1`
	`(1 row)`
		MySQL

inherit_not_null_strict_func	Controls the original **strict** attribute of a function. A function with one parameter can transfer the **NOT NULL** attribute. func(x) is used an example. If func() is the **strict** attribute and x contains the **NOT NULL** constraint, func(x) also contains the **NOT NULL** constraint. The compatible configuration item is effective in some optimization scenarios, for example, **NOT IN** and **COUNT(DISTINCT)** optimization. However, the optimization results may be incorrect in specific scenarios. Currently, this parameter is not configured by default to ensure that the result is correct. However, the performance may be rolled back. If an error occurs, you can set this parameter to roll back to the historical version.	ORA TD MySQL

Configuration Item	Behavior	Applicable Compatibility Mode

disable_compat_minmax_expr_mysql	Specifies the method for processing the input parameter **null** in the **greatest**/**least** expression in MySQL-compatible mode.
	You can configure this parameter to roll back to a historical version.
	● If this parameter is not configured and the input parameter is **null**, **null** is returned.
	`mysql_compatibility_db=# SELECT greatest(1, 2, null), least(1, 2, null);`
	` greatest | least`
	`----------+-------`
	`          |`
	`(1 row)`
	● If this parameter is configured, the maximum or minimum value of non-null parameters is returned.
	`mysql_compatibility_db=# SELECT greatest(1, 2, null), least(1, 2, null);`
	` greatest | least`
	`----------+-------`
	`        2 |     1`
	`(1 row)`
		MySQL

Configuration Item	Behavior	Applicable Compatibility Mode

disable_compat_substr_mysql	Specifies the behavior of the **substr**/**substring** function when the start position pos is ≤ 0 in MySQL-compatible mode.
	You can configure this parameter to roll back to a historical version.
	● If this parameter is not configured, that is, an empty string is returned when pos = 0. When pos < 0, **TRUNCATE** starts from the last |*pos*| character on.
	`mysql_compatibility_db=# SELECT substr('helloworld',0);`
	` substr`
	`--------`
	
	`(1 row)`
	`mysql_compatibility_db=# SELECT substring('helloworld',0),substring('helloworld',-2,4);`
	` substring | substring`
	`-----------+-----------`
	`           | ld`
	`(1 row)`
	● If this parameter is configured and pos is ≤ 0, characters are truncated from the left.
	`mysql_compatibility_db=# SELECT substr('helloworld',0);`
	`   substr`
	`------------`
	` helloworld`
	`(1 row)`
	`mysql_compatibility_db=# SELECT substring('helloworld',0),substring('helloworld',-2,4);`
	` substring  | substring`
	`------------+-----------`
	` helloworld | h`
	`(1 row)`
		MySQL

Configuration Item	Behavior	Applicable Compatibility Mode

disable_compat_trim_mysql	Specifies the method for processing the input parameter in the **trim**/**ltrim**/**rtrim** function in MySQL-compatible mode.
	You can configure this parameter to roll back to a historical version.
	● If this parameter is not configured, the entire substring is matched.
	`mysql_compatibility_db=# SELECT trim('{}{name}{}','{}'),trim('xyznamezyx','xyz');`
	` btrim  |  btrim`
	`--------+---------`
	` {name} | namezyx`
	`(1 row)`
	● If this parameter is configured, a single character in the character set is matched.
	`mysql_compatibility_db=# SELECT trim('{}{name}{}','{}'),trim('xyznamezyx','xyz');`
	` btrim | btrim`
	`-------+-------`
	` name  | name`
	`(1 row)`
		MySQL

light_object_mtime	Specifies whether the **mtime** column in the **pg_object** system catalog records object operations. ● If this parameter is configured, the **GRANT**, **REVOKE**, and **TRUNCATE** operations are not recorded by **mtime**, that is, the **mtime** column is not updated. ● If this parameter is not configured (by default), the **ALTER**, **COMMENT**, **GRANT**, **REVOKE**, and **TRUNCATE** operations are recorded by **mtime**, that is, the **mtime** column is updated.	ORA TD MySQL

Configuration Item	Behavior	Applicable Compatibility Mode

disable_including_all_mysql	In MySQL-compatible mode, this parameter controls whether the **CREATE TABLE...LIKE** syntax is **INCLUDING_ALL**.
	
	By default, this parameter is not set. That is, in MySQL compatibility mode, **CREATE TABLE... LIKE** syntax is **INCLUDING_ALL**.
	
	Set this parameter to roll back to a historical version.
	
	● If this parameter is not set, in MySQL-compatible mode, the **CREATE TABLE... LIKE** syntax is in **INCLUDING_ALL**.
	`mysql_compatibility_db=# CREATE TABLE mysql_like(id int, name varchar(10), score int) distribute by hash(id) COMMENT 'mysql_like';`
	`CREATE TABLE`
	`mysql_compatibility_db=# CREATE index index_like on mysql_like(name);`
	`CREATE INDEX`
	`mysql_compatibility_db=# \d+ mysql_like;`
	`                              Table "public.mysql_like"`
	` Column |          Type          | Modifiers | Storage  | Stats target | Description`
	`--------+-----------------------+-----------+----------+--------------+-------------`
	` id     | integer               |           | plain    |              |`
	` name   | character varying(10) |           | extended |              |`
	` score  | integer               |           | plain    |              |`
	`Indexes:`
	`    "index_like" btree (name) TABLESPACE pg_default`
	`Has OIDs: no`
	`Distribute By: HASH(id)`
	`Location Nodes: ALL DATANODES`
	`Options: orientation=row, compression=no`
	
	`mysql_compatibility_db=# CREATE table copy_like like mysql_like;`
	`CREATE TABLE`
	`mysql_compatibility_db=# \d+ copy_like;`
	`                              Table "public.copy_like"`
	` Column |          Type          | Modifiers | Storage  | Stats target | Description`
	`--------+-----------------------+-----------+----------+--------------+-------------`
	` id     | integer               |           | plain    |              |`
	` name   | character varying(10) |           | extended |              |`
	` score  | integer               |           | plain    |              |`
	`Indexes:`
	`    "copy_like_name_idx" btree (name) TABLESPACE pg_default`
	`Has OIDs: no`
	`Distribute By: HASH(id)`
	`Location Nodes: ALL DATANODES`
	`Options: orientation=row, compression=no`
	
	● If this parameter is set, in MySQL-compatible mode, the **CREATE TABLE... LIKE** syntax is empty.
		MySQL

Configuration Item	Behavior	Applicable Compatibility Mode

	`mysql_compatibility_db=# SET behavior_compat_options = 'disable_including_all_mysql';`
	`SET`
	`mysql_compatibility_db=# CREATE TABLE mysql_copy like mysql_like;`
	`NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using round-robin as the distribution mode by default.`
	`HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.`
	`CREATE TABLE`
	`mysql_db=# \d+ mysql_copy;`
	`                             Table "public.mysql_copy"`
	` Column |         Type          | Modifiers | Storage  | Stats target | Description`
	`--------+-----------------------+-----------+----------+--------------+-------------`
	` id     | integer               |           | plain    |              |`
	` name   | character varying(10) |           | extended |              |`
	` score  | integer               |           | plain    |              |`
	`Has OIDs: no`
	`Distribute By: ROUND ROBIN`
	`Location Nodes: ALL DATANODES`
	`Options: orientation=row, compression=no`	

cte_onetime_inline	Indicates whether to execute **inline** for non-stream plans. ● When this parameter is set, the CTE that is not in a stream plan and is referenced only once executes **inline**. ● If this parameter is not set, the CTE that is not in a stream plan and is referenced only once does not execute **inline**.	ORA TD MySQL

skip_first_after_mysql	Determines whether to ignore the **FIRST/AFTER colname** syntax in **ALTER TABLE ADD/MODIFY/CHANGE COLUMN** in MySQL compatibility mode.
	● If this parameter is set, the **FIRST/AFTER colname** syntax is ignored and executing this syntax does not cause errors
	`mysql_compatibility_db=# SET behavior_compat_options = 'skip_first_after_mysql';`
	`mysql_compatibility_db=# ALTER TABLE t1 add column b text after a;`
	`ALTER TABLE`
	● If this parameter is not set, the **FIRST/AFTER colname** syntax is not supported, and executing this syntax causes error.
	`mysql_compatibility_db=# SET behavior_compat_options = '';`
	`mysql_compatibility_db=# ALTER TABLE t1 add column b text after a;`
	`ERROR:  FIRST/AFTER is not yet supported.`
		MySQL

Configuration Item	Behavior	Applicable Compatibility Mode

enable_division_by_zero_mysql	Specifies whether to report an error when the divisor is 0 in MySQL compatibility mode. (This configuration item is supported only by clusters of 8.1.3.110 and later versions.)
	● If this parameter is set, NULL is returned if the divisor is 0 in a division or modulo operation.
	`compatible_mysql_db=# SET behavior_compat_options = 'enable_division_by_zero_mysql';`
	`SET`
	`compatible_mysql_db=# SELECT 1/0 as test;`
	` test`
	`----------`
	
	`(1 row)`
	● If this parameter is not set, an error is returned if the divisor is 0 in a division or modulo operation.
	`compatible_mysql_db=# SELECT 1/0;`
	`ERROR:  division by zero`
		MySQL

merge_into_with_trigger	Controls whether the MERGE INTO operation can be performed on tables with triggers. (This parameter is supported only in 8.1.3.200 and later cluster versions.) ● When this option is set, the MERGE INTO operation can be performed on tables with triggers. When the MERGE INTO operation is performed, the trigger on the table is not activated. ● If this option is not set, an error is reported when the MERGE INTO operation is performed on a table with triggers.	ORA TD MySQL
add_column_default_v_func	Controls whether **expression** in **alter table add column default expression** supports volatile functions. (This parameter is supported only in 8.1.3.200 and later cluster versions.) ● If this option is selected, **expression** in **alter table add column default expression** supports volatile functions. ● If this option is not selected, **expression** in **alter table add column default expression** does not support volatile functions. If **expression** contains volatile functions, an error will be reported during statement execution.	ORA TD MySQL

Configuration Item	Behavior	Applicable Compatibility Mode

disable_gc_fdw_filter_partial_pushdown	Controls whether filter criteria are pushed down when filter criteria are used to query data in a collaborative analysis foreign table (type: gc_fdw). (This parameter is supported only in 8.1.3.310 and later cluster versions.)
	● When this option is selected, if the filter criteria contain elements (such as non-immutable functions) that do not meet the pushdown conditions, all filter criteria are not pushed down to ensure the normal generation of the result set document. This behavior is compatible with the behavior in versions earlier than 8.1.3.310.
	`-- Create a table in the source cluster.`
	`CREATE TABLE t1(c1 INT, c2 INT, c3 INT) DISTRIBUTE BY HASH(c1);`
	`-- Create a foreign table with the same structure in the local cluster.`
	`CREATE SERVER server_remote FOREIGN DATA WRAPPER gc_fdw options(ADDRESS 'address', DBNAME 'dbname', USERNAME 'username', PASSWORD 'password');`
	`CREATE FOREIGN TABLE t1(c1 INT, c2 INT, c3 INT) SERVER server_remote;`
	`-- Enable the parameter and see the pushdown behavior.`
	`SET behavior_compat_options = 'disable_gc_fdw_filter_partial_pushdown';`
	`EXPLAIN (verbose on,costs off) SELECT * FROM t1 WHERE c1>3 AND c2 <100 AND now() - '20230101' < c3;`
	`QUERY PLAN`
	`------------------------------------------------------------------------------------------------------------------------------------------------------`
	` Streaming (type: GATHER)`
	`   Output: c1, c2, c3`
	`   Node/s: All datanodes`
	`   ->  Foreign Scan on ca_schema.t1`
	`         Output: c1, c2, c3`
	`         Filter: ((t1.c1 > 3) AND (t1.c2 < 100) AND ((now() - '2023-01-01 00:00:00-08'::timestamp with time zone) < (t1.c3)::interval))`
	`         Remote SQL: SELECT c1, c2, c3 FROM ca_schema.t1`
	`(7 rows)`
	● If this parameter is not set, the filter criteria that can be pushed down are executed in the source cluster, and the filter criteria that cannot be pushed down are executed in the local cluster. This improves the query efficiency of foreign tables.
	`-- Disable this parameter and see the pushdown behavior.`
	`SET behavior_compat_options = '';`
	`EXPLAIN (verbose on,costs off) SELECT * FROM t1 WHERE c1>3 AND c2 <100 AND now() - '20230101' < c3;`
	`QUERY`
		ORA
		TD
		MySQL

Configuration Item	Behavior	Applicable Compatibility Mode
	```
	PLAN
	-------------------------------------------------------------------------------------------------------
	 Streaming (type: GATHER)
	   Output: c1, c2, c3
	   Node/s: All datanodes
	   ->  Foreign Scan on ca_schema.t1
	         Output: c1, c2, c3
	         Filter: ((now() - '2023-01-01 00:00:00-08'::timestamp with time zone) < (t1.c3)::interval)
	         Remote SQL: SELECT c1, c2, c3 FROM ca_schema.t1 WHERE ((c1 > 3)) AND ((c2 < 100))
	(7 rows)
	```

normalize_negative_zero	Controls whether the **ceil()** and **round()** functions will produce a negative zero when dealing with certain float values. This parameter is supported only by clusters of version 8.1.3.333 and later.
	● When this parameter is set, **ceil()** processes **(-1,0)** and **round()** processes **[-0.5, 0)**. The return value is **0**.
	```
	SET behavior_compat_options='normalize_negative_zero';
	SELECT ceil(cast(-0.1 as float));
	 ceil
	------
	    0
	(1 row)
	SELECT round(cast(-0.1 as FLOAT));
	 round
	-------
	     0
	(1 row)
	```
	● If this parameter is not set, **-0** is returned when **ceil()** processes **(-1,0)** and **round()** processes **[-0.5, 0)**.
	```
	SET behavior_compat_options = '';
	SELECT ceil(cast(-0.1 as FLOAT));
	 ceil
	------
	   -0
	(1 row)
	SELECT round(cast(-0.1 as FLOAT));
	 round
	-------
	    -0
	(1 row)
	```
		ORA
		TD
		MySQL

Configuration Item	Behavior	Applicable Compatibility Mode
disable_client_detection_commit	Specifies whether to verify the client connection before committing each transaction. If the connection is not present, an error will be reported, and the transaction will be rolled back to prevent duplicate data delivery due to disconnection. This parameter is supported only by clusters of version 8.1.3.333 and later. ● If this parameter is not set, the system will verify the client connection before committing each transaction. ● If this parameter is set, the system will not verify the client connection before committing each transaction.	ORA TD MySQL
enable_trunc_orc_string	Controls the foreign table query behavior when the foreign table field is in ORC format and the data type is varchar(n), but the field type in the ORC file is string and the length of the string exceeds n. This parameter is available only in clusters of version 8.1.3.336, 8.3.0.100, 910.100, and later. ● If this parameter is not set, an error message is returned, indicating that the field is too long. ● If this parameter is set, the query is responded to, and the result is truncated by the length defined by varchar(n).	ORA TD MySQL
gds_fill_multi_missing_fields	Controls the behavior when the GDS foreign table fault tolerance parameter **fill_missing_fields** is set to **true** or **on**. When **fill_missing_fields** is set to **true** or **on** in a GDS foreign table, any missing columns at the end of a row in the data source file are automatically set to **NULL**. Before this, only the last column in a row of the data source file can be missing without an error being reported. This parameter is available only in clusters of version 8.1.3.336, 8.2.1.200, 9.1.0.100, and later. ● If this option is specified, the GDS foreign table tolerates the missing of multiple last columns in a row of the source data file. ● If this option is not specified, only the missing of the last column in a row of the data source file is tolerated in the GDS foreign table. This parameter compatible with historical behavior.	ORA TD MySQL

internal_compat_options

Parameter description: Specifies database compatibility behavior. Multiple items are separated by commas (,). This parameter is supported only by clusters of version 8.1.3.333 and later.

Type: SIGHUP

Value range: a string

Default value: In upgrade scenarios, the default value of this parameter is the same as that in the cluster before the upgrade. In a cluster installation scenario, the default value of this parameter is empty.

Table 15-7 Compatibility configuration items

Configuration Item	Behavior
light_proxy_permission_compat	Nested query permission configuration item in the light proxy scenario. ● If this parameter is not set, you must have the query permission for nested queries in the light proxy scenario. ● Enabling this parameter allows for nested queries in the light proxy scenario, regardless of permissions.

redact_compat_options

Parameter description: Specifies the compatibility option for calculation using masked data. This parameter is supported by version 8.1.3 or later clusters.

Type: USERSET

Value range: a string

- **none** indicates that compatibility options are specified.
- **disable_comparison_operator_mask** indicates that comparison operators that do not expose raw data can bypass the data masking check and generate the actual calculation result.

Default value: **none**

table_skewness_warning_threshold

Parameter description: Specifies the threshold for triggering a table skew alarm.

Type: SUSET

Value range: a floating point number ranging from 0 to 1

Default value: **1**

table_skewness_warning_rows

Parameter description: Specifies the minimum number of rows for triggering a table skew alarm.

Type: SUSET

Value range: an integer ranging from **0** to **INT_MAX**

Default value: **100000**

auto_process_residualfile

Parameter description: Specifies whether to enable the residual file recording function.

Type: SIGHUP

Value range: Boolean

- **on** indicates that the residual file recording function is enabled.
- **off** indicates that the residual file recording function is disabled.

Default value: **off**

enable_view_update

Parameter description: Enables the view update function or not.

Type: POSTMASTER

Value range: Boolean

- **on** indicates that the view update function is enabled.
- **off** indicates that the view update function is disabled.

Default value: **off**

view_independent

Parameter description: Decouples views from tables, functions, and synonyms or not. After the base table is restored, automatic association and re-creation are supported.

Type: SIGHUP

Value range: Boolean

- **on** indicates that the view decoupling function is enabled. Tables, functions, synonyms, and other views on which views depend can be deleted separately (except temporary tables and temporary views). Associated views are reserved but unavailable.
- **off** indicates that the view decoupling function is disabled. Tables, functions, synonyms, and other views on which views depend cannot be deleted separately. You can only delete them in the cascade mode.

Default value: **off**

bulkload_report_threshold

Parameter description: Sets the threshold for reporting import and export statistics. When the data volume exceeds this threshold, the **PGXC_BULKLOAD_STATISTICS** view can be used to query synchronized data volume, record count, execution time, and other information.

Type: SIGHUP

Value range: an integer ranging from **0** to **INT_MAX**

Default value: **50**

assign_abort_xid

Parameter description: Determines the transaction to be aborted based on the specified XID in a query.

Type: USERSET

Value range: a character string with the specified XID

CAUTION

This parameter is used only for quick restoration if a user deletes data by mistake (DELETE operation). Do not use this parameter in other scenarios. Otherwise, visible transaction errors may occur.

default_distribution_mode

Parameter description: Specifies the default distribution mode of a table. This feature is supported only in 8.1.2 or later.

Type: USERSET

Value range: enumerated values

- **roundrobin**: If the distribution mode is not specified during table creation, the default distribution mode is selected according to the following rules:
  a. If the primary key or unique constraint is included during table creation, hash distribution is selected. The distribution column is the column corresponding to the primary key or unique constraint.
  b. If the primary key or unique constraint is not included during table creation, round-robin distribution is selected.
- **hash**: If the distribution mode is not specified during table creation, the default distribution mode is selected according to the following rules:
  a. If the primary key or unique constraint is included during table creation, hash distribution is selected. The distribution column is the column corresponding to the primary key or unique constraint.
  b. If the primary key or unique constraint is not included during table creation but there are columns whose data types can be used as distribution columns, hash distribution is selected. The distribution

column is the first column whose data type can be used as a distribution column.

c. If the primary key or unique constraint is not included during table creation and no column whose data type can be used as a distribution column exists, round-robin distribution is selected.

Default value: **roundrobin**

NOTE

The default value of this parameter is **roundrobin** for a new GaussDB(DWS) 8.1.2 cluster and is **hash** for an upgrade to GaussDB(DWS) 8.1.2.

16 GaussDB(DWS) Developer Terms

Term	Description
A – E	
ACID	Atomicity, Consistency, Isolation, and Durability (ACID). These are a set of properties of database transactions in a DBMS.
cluster ring	A cluster ring consists of several physical servers. The primary-standby-secondary relationships among its DNs do not involve external DNs. That is, none of the primary, standby, or secondary counterparts of DNs belonging to the ring are deployed in other rings. A ring is the smallest unit used for scaling.
Bgwriter	A background write thread created when the database starts. The thread pushes dirty pages in the database to a permanent device (such as a disk).
bit	The smallest unit of information handled by a computer. One bit is expressed as a 1 or a 0 in a binary numeral, or as a true or a false logical condition. A bit is physically represented by an element such as high or low voltage at one point in a circuit, or a small spot on a disk that is magnetized in one way or the other. A single bit conveys little information a human would consider meaningful. A group of eight bits, however, makes up a byte, which can be used to represent many types of information, such as a letter of the alphabet, a decimal digit, or other character.
Bloom filter	Bloom filter is a space-efficient binary vectorized data structure, conceived by Burton Howard Bloom in 1970, that is used to test whether an element is a member of a set. False positive matches are possible, but false negatives are not, in other words, a query returns either "possibly in set (possible error)" or "definitely not in set". In the cases, Bloom filter sacrificed the accuracy for time and space.

Term	Description
CCN	The Central Coordinator (CCN) is a node responsible for determining, queuing, and scheduling complex operations in each CN to enable the dynamic load management of GaussDB(DWS).
CIDR	Classless Inter-Domain Routing (CIDR). CIDR abandons the traditional class-based (class A: 8; class B: 16; and class C: 24) address allocation mode and allows the use of address prefixes of any length, effectively improving the utilization of address space. A CIDR address is in the format of *IP address*/*Number of bits in a network ID*. For example, in **192.168.23.35/21**, **21** indicates that the first 21 bits are the network prefix and others are the host ID.
Cgroups	A control group (Cgroup), also called a priority group (PG) in GaussDB(DWS). The Cgroup is a kernel feature of SUSE Linux and Red Hat that can limit, account for, and isolate the resource usage of a collection of processes.
CLI	Command-line interface (CLI). Users use the CLI to interact with applications. Its input and output are based on texts. Commands are entered through keyboards or similar devices and are compiled and executed by applications. The results are displayed in text or graphic forms on the terminal interface.
CM	Cluster Manager (CM) manages and monitors the running status of functional units and physical resources in the distributed system, ensuring stable running of the entire system.
CMS	The Cluster Management Service (CMS) component manages the cluster status.
CN	The Coordinator (CN) stores database metadata, splits query tasks and supports their execution, and aggregates the query results returned from DNs.
CU	Compression Unit (CU) is the smallest storage unit in a column-storage table.
core file	A file that is created when memory overwriting, assertion failures, or access to invalid memory occurs in a process, causing it to fail. This file is then used for further analysis. A core file contains a memory dump, in an all-binary and port-specific format. The name of a core file consists of the word "core" and the OS process ID. The core file is available regardless of the type of platform.

Term	Description
core dump	When a program stops abnormally, the core dump, memory dump, or system dump records the state of the working memory of the program at that point in time. In practice, other key pieces of program state are usually dumped at the same time, including the processor registers, which may include the program counter and stack pointer, memory management information, and other processor and OS flags and information. A core dump is often used to assist diagnosis and computer program debugging.
DBA	A database administrator (DBA) instructs or executes database maintenance operations.
DBLINK	An object defining the path from one database to another. A remote database object can be queried with DBLINK.
DBMS	Database Management System (DBMS) is a piece of system management software that allows users to access information in a database. This is a collection of programs that allows you to access, manage, and query data in a database. A DBMS can be classified as memory DBMS or disk DBMS based on the location of the data.
DCL	Data control language (DCL)
DDL	Data definition language (DDL)
DML	Data manipulation language (DML)
DN	Datanode performs table data storage and query operations.
ETCD	The Editable Text Configuration Daemon (ETCD) is a distributed key-value storage system used for configuration sharing and service discovery (registration and search).
ETL	Extract-Transform-Load (ETL) refers to the process of data transmission from the source to the target database.
Extension Connector	Extension Connector is provided by GaussDB(DWS) to process data across clusters. It can send SQL statements to Spark, and can return execution results to your database.
Backup	A backup, or the process of backing up, refers to the copying and archiving of computer data in case of data loss.
backup and restoration	A collection of concepts, procedures, and strategies to protect data loss caused by invalid media or misoperations.
standby server	A node in the GaussDB(DWS) HA solution. It functions as a backup of the primary server. If the primary server is behaving abnormally, the standby server is promoted to primary, ensuring data service continuity.

Term	Description
crash	A crash (or system crash) is an event in which a computer or a program (such as a software application or an OS) ceases to function properly. Often the program will exit after encountering this type of error. Sometimes the offending program may appear to freeze or hang until a crash reporting service documents details of the crash. If the program is a critical part of the OS kernel, the entire computer may crash (possibly resulting in a fatal system error).
encoding	Encoding is representing data and information using code so that it can be processed and analyzed by a computer. Characters, digits, and other objects can be converted into digital code, or information and data can be converted into the required electrical pulse signals based on predefined rules.
encoding technology	A technology that presents data using a specific set of characters, which can be identified by computer hardware and software.
table	A set of columns and rows. Each column is referred to as a field. The value in each field represents a data type. For example, if a table contains people's names, cities, and states, it has three columns: **Name**, **City**, and **State**. In every row in the table, the **Name** column contains a name, the **City** column contains a city, and the **State** column contains a state.
tablespace	A tablespace is a logical storage structure that contains tables, indexes, large objects, and long data. A tablespace provides an abstract layer between physical data and logical data, and provides storage space for all database objects. When you create a table, you can specify which tablespace it belongs to.
concurrency control	A DBMS service that ensures data integrity when multiple transactions are concurrently executed in a multi-user environment. In a multi-threaded environment, GaussDB(DWS) concurrency control ensures that database operations are safe and all database transactions remain consistent at any given time.
query	Specifies requests sent to the database, such as updating, modifying, querying, or deleting information.
query operator	An iterator or a query tree node, which is a basic unit for the execution of a query. Execution of a query can be split into one or more query operators. Common query operators include scan, join, and aggregation.
query fragment	Each query task can be split into one or more query fragments. Each query fragment consists of one or more query operators and can independently run on a node. Query fragments exchange data through data flow operators.

Term	Description
durability	One of the ACID features of database transactions. Durability indicates that transactions that have been committed will permanently survive and not be rolled back.
stored procedure	A group of SQL statements compiled into a single execution plan and stored in a large database system. Users can specify a name and parameters (if any) for a stored procedure to execute the procedure.
OS	An operating system (OS) is loaded by a bootstrap program to a computer to manage other programs in the computer. Other programs are applications or application programs.
secondary server	To ensure high cluster availability, the primary server synchronizes logs to the secondary server if data synchronization between the primary and standby servers fails. If the primary server suddenly breaks down, the standby server is promoted to primary and synchronizes logs from the secondary server for the duration of the breakdown.
BLOB	Binary large object (BLOB) is a collection of binary data stored in a database, such as videos, audio, and images.
dynamic load balancing	In GaussDB(DWS), dynamic load balancing automatically adjusts the number of concurrent jobs based on the usage of CPU, I/O, and memory to avoid service errors and to prevent the system from stop responding due to system overload.
segment	A segment in the database indicates a part containing one or more regions. Region is the smallest range of a database and consists of data blocks. One or more segments comprise a tablespace.
F – J	
failover	Automatic switchover from a faulty node to its standby node. Reversely, automatic switchback from the standby node to the primary node is called failback.
FDW	A foreign data wrapper (FDW) is a SQL interface provided by Postgres. It is used to access big data objects stored in remote data so that DBAs can integrate data from unrelated data sources and store them in public schema in the database.

Term	Description
freeze	An operation automatically performed by the AutoVacuum Worker process when transaction IDs are exhausted. GaussDB(DWS) records transaction IDs in row headings. When a transaction reads a row, the transaction ID in the row heading and the actual transaction ID are compared to determine whether this row is explicit. Transaction IDs are integers containing no symbols. If exhausted, transaction IDs are re-calculated outside of the integer range, causing the explicit rows to become implicit. To prevent such a problem, the freeze operation marks a transaction ID as a special ID. Rows marked with these special transaction IDs are explicit to all transactions.
GDB	As a GNU debugger, GDB allows you to see what is going on 'inside' another program while it executes or what another program was doing the moment that it crashed. GDB can perform four main kinds of things (make PDK functions stronger) to help you catch bugs in the act: ● Starts your program, specifying anything that might affect its behavior. ● Stops a program in a specific condition. ● Checks what happens when a program stops. ● Modifies the program content to rectify the fault and proceeds with the next one.
GDS	General Data Service (GDS). To import data to GaussDB(DWS), you need to deploy the tool on the server where the source data is stored so that DNs can use this tool to obtain data.
GIN index	Generalized inverted index (GIN) is used 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.
GNU	The GNU Project was publicly announced on September 27, 1983 by Richard Stallman, aiming at building an OS composed wholly of free software. GNU is a recursive acronym for "GNU's Not Unix!". Stallman announced that GNU should be pronounced as Guh-NOO. Technically, GNU is similar to Unix in design, a widely used commercial OS. However, GNU is free software and contains no Unix code.
gsql	GaussDB(DWS) interaction terminal. It enables you to interactively type in queries, issue them to GaussDB(DWS), and view the query results. Queries can also be entered from files. **gsql** supports many meta commands and shell-like commands, allowing you to conveniently compile scripts and automate tasks.
GTM	Global Transaction Manager (GTM) manages the status of transactions.

Term	Description
GUC	Grand unified configuration (GUC) includes parameters for running databases, the values of which determine database system behavior.
HA	High availability (HA) is a solution in which two modules operate in primary/standby mode to achieve high availability. This solution helps to minimize the duration of service interruptions caused by routine maintenance (planned) or sudden system breakdowns (unplanned), improving the system and application usability.
HBA	Host-based authentication (HBA) allows hosts to authenticate on behalf of all or some of the system users. It can apply to all users on a system or a subset using the **Match** directive. This type of authentication can be useful for managing computing clusters and other fairly homogenous pools of machines. In all, three files on the server and one on the client must be modified to prepare for host-based authentication.
HDFS	Hadoop Distributed File System (HDFS) is a subproject of Apache Hadoop. HDFS is highly fault tolerant and is designed to run on low-end hardware. The HDFS provides high-throughput access to large data sets and is ideal for applications having large data sets.
server	A combination of hardware and software designed for providing clients with services. This word alone refers to the computer running the server OS, or the software or dedicated hardware providing services.
advanced package	Logical and functional stored procedures and functions provided by GaussDB(DWS).
isolation	One of the ACID features of database transactions. Isolation means that the operations inside a transaction and data used are isolated from other concurrent transactions. The concurrent transactions do not affect each other.
relational database	A database created using a relational model. It processes data using methods of set algebra.
archive thread	A thread started when the archive function is enabled on a database. The thread archives database logs to a specified path.
failover	The automatic substitution of a functionally equivalent system component for a failed one. The system component can be a processor, server, network, or database.
environment variable	An environment variable defines the part of the environment in which a process runs. For example, it can define the part of the environment as the main directory, command search path, terminal that is in use, or the current time zone.

Term	Description
checkpoint	A mechanism that stores data in the database memory to disks at a certain time. GaussDB(DWS) periodically stores the data of committed and uncommitted transactions to disks. The data and redo logs can be used for database restoration if a database restarts or breaks down.
encryption	A function hiding information content during data transmission to prevent the unauthorized use of the information.
node	Cluster nodes (or nodes) are physical and virtual severs that make up the GaussDB(DWS) cluster environment.
error correction	A technique that automatically detects and corrects errors in software and data streams to improve system stability and reliability.
process	An instance of a computer program that is being executed. A process may be made up of multiple threads of execution. Other processes cannot use a thread occupied by the process.
PITR	Point-In-Time Recovery (PITR) is a backup and restoration feature of GaussDB(DWS). Data can be restored to a specified point in time if backup data and WAL logs are normal.
record	In a relational database, a record corresponds to data in each row of a table.
cluster	A cluster is an independent system consisting of servers and other resources, ensuring high availability. In certain conditions, clusters can implement load balancing and concurrent processing of transactions.
K – O	
LLVM	LLVM is short for Low Level Virtual Machine. Low Level Virtual Machine (LLVM) is a compiler framework written in C++ and is designed to optimize the compile-time, link-time, run-time, and idle-time of programs that are written in arbitrary programming languages. It is open to developers and compatible with existing scripts. GaussDB(DWS) LLVM dynamic compilation can be used to generate customized machine code for each query to replace original common functions. Query performance is improved by reducing redundant judgment conditions and virtual function invocation, and by making local data more accurate during actual queries.
LVS	Linux Virtual Server (LVS), a virtual server cluster system, is used for balancing the load of a cluster.

Term	Description
logical replication	Data synchronization mode between primary and standby databases or between two clusters. Different from physical replication which replays physical logs, logical replication transfers logical logs between two clusters or synchronizes data through SQL statements in logical logs.
logical log	Logs recording database changes made through SQL statements. Generally, the changes are logged at the row level. Logical logs are different from physical logs that record changes of physical pages.
logical decoding	Logic decoding is a process of extracting all permanent changes in database tables into a clear and easy-to-understand format by decoding Xlogs.
logical replication slot	In a logical replication process, logic replication slots are used to prevent Xlogs from being reclaimed by the system or **VACUUM**. In GaussDB(DWS), a logical replication slot is an object that records logical decoding positions. It can be created, deleted, read, and pushed by invoking SQL functions.
MPP	Massive Parallel Processing (MPP) refers to cluster architecture that consists of multiple machines. The architecture is also called a cluster system.
MVCC	Multi-Version Concurrency Control (MVCC) is a protocol that allows a tuple to have multiple versions, on which different query operations can be performed. A basic advantage is that read and write operations do not conflict.
NameNode	The NameNode is the centerpiece of a Hadoop file system, managing the namespace of the file system and client access to files.
Node Group	In GaussDB(DWS), a Node Group refers to a DN set, which is a sub-cluster. Node Groups can be classified into Storage Node Groups, which store local table data; and Computing Node Groups, which perform aggregation and join for queries.
OLAP	Online analytical processing (OLAP) is the most important application in the database warehouse system. It is dedicated to complex analytical operations, helps decision makers and executives to make decisions, and rapidly and flexibly processes complex queries involving a great amount of data based on analysts' requirements. In addition, the OLAP provides decision makers with query results that are easy to understand, allowing them to learn the operating status of the enterprise. These decision makers can then produce informed and accurate solutions based on the query results.
OM	Operations Management (OM) provides management interfaces and tools for routine maintenance and configuration management of the cluster.

Term	Description
ORC	Optimized Row Columnar (ORC) is a widely used file format for structured data in a Hadoop system. It was introduced from the Hadoop HIVE project.
client	A computer or program that accesses or requests services from another computer or program.
free space management	A mechanism for managing free space in a table. This mechanism enables the database system to record free space in each table and establish an easy-to-search data structure, accelerating operations (such as INSERT) performed on the free space.
cross-cluster	In GaussDB(DWS), users can access data in other DBMS through foreign tables or using an Extension Connector. Such access is cross-cluster.
junk tuple	A tuple that is deleted using the **DELETE** and **UPDATE** statements. When deleting a tuple, GaussDB(DWS) only marks the tuples that are to be cleared. The Vacuum thread will then periodically clear these junk tuples.
column	An equivalent concept of "field". A database table consists of one or more columns. Together they describe all attributes of a record in the table.
logical node	Multiple logical nodes can be installed on the same node. A logical node is a database instance.
schema	Collection of database objects, including logical structures, such as tables, views, sequences, stored procedures, synonyms, indexes, clusters, and database links.
schema file	A SQL file that determines the database structure.
P – T	
Page	Minimum memory unit for row storage in the GaussDB(DWS) relational object structure. The default size of a page is 8 KB.
PostgreSQL	An open-source DBMS developed by volunteers all over the world. PostgreSQL is not controlled by any companies or individuals. Its source code can be used for free.
Postgres-XC	Postgres-XC is an open source PostgreSQL cluster to provide write-scalable, synchronous, multi-master PostgreSQL cluster solution.
Postmaster	A thread started when the database service is started. It listens to connection requests from other nodes in the cluster or from clients. After receiving and accepting a connection request from the standby server, the primary server creates a WAL Sender thread to interact with the standby server.

Term	Description
RHEL	Red Hat Enterprise Linux (RHEL)
redo log	A log that contains information required for performing an operation again in a database. If a database is faulty, redo logs can be used to restore the database to its original state.
SCTP	The Stream Control Transmission Protocol (SCTP) is a transport-layer protocol defined by Internet Engineering Task Force (IETF) in 2000. The protocol ensures the reliability of datagram transport based on unreliable service transmission protocols by transferring SCN narrowband signaling over IP network.
savepoint	A savepoint marks the end of a sub-transaction (also known as a nested transaction) in a relational DBMS. The process of a long transaction can be divided into several parts. After a part is successfully executed, a savepoint will be created. If later execution fails, the transaction will be rolled back to the savepoint instead of being totally rolled back. This is helpful for recovering database applications from complicated errors. If an error occurs in a multi-statement transaction, the application can possibly recover by rolling back to the save point without terminating the entire transaction.
session	A task created by a database for a connection when an application attempts to connect to the database. Sessions are managed by the session manager. They execute initial tasks to perform all user operations.
shared-nothing architecture	A distributed computing architecture, in which none of the nodes share CPUs or storage resources. This architecture has good scalability.
SLES	SUSE Linux Enterprise Server (SLES) is an enterprise Linux OS provided by SUSE.
SMP	Symmetric multiprocessing (SMP) lets multiple CPUs run on a computer and share the same memory and bus. To ensure an SMP system achieves high performance, an OS must support multi-tasking and multi-thread processing. In databases, SMP means to concurrently execute queries using the multi-thread technology, efficiently using all CPU resources and improving query performance.
SQL	Structure Query Language (SQL) is a standard database query language. It consists of DDL, DML, and DCL.

Term	Description
SSL	Secure Socket Layer (SSL) is a network security protocol introduced by Netscape. SSL is a security protocol based on the TCP and IP communications protocols and uses the public key technology. SSL supports a wide range of networks and provides three basic security services, all of which use the public key technology. SSL ensures the security of service communication through the network by establishing a secure connection between the client and server and then sending data through this connection.
convergence ratio	Downlink to uplink bandwidth ratio of a switch. A high convergence ratio indicates a highly converged traffic environment and severe packet loss.
TCP	Transmission Control Protocol (TCP) sends and receives data through the IP protocol. It splits data into packets for sending, and checks and reassembles received package to obtain original information. TCP is a connection-oriented, reliable protocol that ensures information correctness in transmission.
trace	A way of logging to record information about the way a program is executed. This information is typically used by programmers for debugging purposes. System administrators and technical support can diagnose common problems by using software monitoring tools and based on this information.
full backup	Backup of the entire database cluster.
full synchronization	A data synchronization mechanism specified in the GaussDB(DWS) HA solution. Used to synchronize all data from the primary server to a standby server.
Log File	A file to which a computer system writes a record of its activities.
transaction	A logical unit of work performed within a DBMS against a database. A transaction consists of a limited database operation sequence, and must have ACID features.
data	A representation of facts or directives for manual or automatic communication, explanation, or processing. Data includes constants, variables, arrays, and strings.
data redistribution	A process whereby a data table is redistributed among nodes after users change the data distribution mode.

Term	Description
data distribution	A mode in which table data is split and stored on each database instance in a distributed system. Table data can be distributed in hash, replication, or random mode. In hash mode, a hash value is calculated based on the value of a specified column in a tuple, and then the target storage location of the tuple is determined based on the mapping between nodes and hash values. In replication mode, tuples are replicated to all nodes. In random mode, data is randomly distributed to the nodes.
data partitioning	A division of a logical database or its constituent elements into multiple parts (partitions) whose data does not overlap based on specified ranges. Data is mapped to storage locations based on the value ranges of specific columns in a tuple.
Database Name	A collection of data that is stored together and can be accessed, managed, and updated. Data in a view in the database can be classified into the following types: numerals, full text, digits, and images.
DB instance	A database instance consists of a process in GaussDB(DWS) and files controlled by the process. GaussDB(DWS) installs multiple database instances on one physical node. GTM, CM, CN, and DN installed on cluster nodes are all database instances. A database instance is also called a logical node.
database HA	GaussDB(DWS) provides a highly reliable HA solution. Every logical node in GaussDB(DWS) is identified as a primary or standby node. Only one GaussDB(DWS) node is identified as primary at a time. When the HA system is deployed for the first time, the primary server synchronizes all data from each standby server (full synchronization). The HA system then synchronizes only data that is new or has been modified from each standby server (incremental synchronization). When the HA system is running, the primary server can receive data read and write operation requests and the standby servers only synchronize logs.
database file	A binary file that stores user data and the data inside the database system.
data flow operator	An operator that exchanges data among query fragments. By their input/output relationships, data flows can be categorized into Gather flows, Broadcast flows, and Redistribution flows. **Gather** combines multiple query fragments of data into one. Broadcast forwards the data of one query fragment to multiple query fragments. **Redistribution** reorganizes the data of multiple query fragments and then redistributes the reorganized data to multiple query fragments.

Term	Description
data dictionary	A reserved table within a database which is used to store information about the database itself. The information includes database design information, stored procedure information, user rights, user statistics, database process information, database increase statistics, and database performance statistics.
deadlock	Unresolved contention for the use of resources.
index	An ordered data structure in the database management system. An index accelerates querying and the updating of data in database tables.
statistics	Information that is automatically collected by databases, including table-level information (number of tuples and number of pages) and column-level information (column value range distribution histogram). Statistics in databases are used to estimate the cost of execution plans to find the plan with the lowest cost.
stop word	In computing, stop words are words which are filtered out before or after processing of natural language data (text), saving storage space and improving search efficiency.
U – Z	
vacuum	A thread that is periodically started up by a database to clear junk tuples. Multiple Vacuum threads can be started concurrently by setting a parameter.
verbose	The VERBOSE option specifies the information to be displayed.
WAL	Write-ahead logging (WAL) is a standard method for logging a transaction. Corresponding logs must be written into a permanent device before a data file (carrier for a table and index) is modified.
WAL Receiver	A thread created by the standby server during database duplication. The thread is used to receive data and commands from the primary server and to tell the primary server that the data and commands have been acknowledged. Only one WAL receiver thread can run on one standby server.
WAL Sender	A thread created on the primary server when the primary server has received a connection request from a standby server during database replication. This thread is used to send data and commands to standby servers and to receive responses from the standby servers. Multiple WAL Sender threads may run on one primary server. Each WAL Sender thread corresponds to a connection request initiated by a standby server.
WAL Writer	A thread for writing redo logs that are created when a database is started. This thread is used to write logs in the memory to a permanent device, such as a disk.

Term	Description
WLM	The WorkLoad Manager (WLM) is a module for controlling and allocating system resources in GaussDB(DWS).
Xlog	A transaction log. A logical node can have only one Xlog file.
xDR	X detailed record. It refers to detailed records on the user and signaling plans and can be categorized into charging data records (CDRs), user flow data records (UFDRs), transaction detail records (TDRs), and data records (SDRs).
network backup	Network backup provides a comprehensive and flexible data protection solution to MS Windows, UNIX, and Linux platforms. Network backup can back up, archive, and restore files, folders, directories, volumes, and partitions on a computer.
physical node	A physical machine or device.
system catalog	A table storing meta information about the database. The meta information includes user tables, indexes, columns, functions, and the data types in a database.
pushdown	GaussDB(DWS) is a distributed database, where CN can send a query plan to multiple DNs for parallel execution. This CN behavior is called pushdown. It achieves better query performance than extracting data to CN for query.
compression	Data compression, source coding, or bit-rate reduction involves encoding information that uses fewer bits than the original representation. Compression can be either lossy or lossless. Lossless compression reduces bits by identifying and eliminating statistical redundancy. No information is lost in lossless compression. Lossy compression reduces bits by identifying and removing unnecessary or unimportant information. The process of reducing the size of a data file is commonly referred as data compression, although its formal name is source coding (coding done at the source of the data, before it is stored or transmitted).
consistency	One of the ACID features of database transactions. Consistency is a database status. In such a status, data in the database must comply with integrity constraints.
metadata	Data that provides information about other data. Metadata describes the source, size, format, or other characteristics of data. In database columns, metadata explains the content of a data warehouse.

Term	Description
atomicity	One of the ACID features of database transactions. Atomicity means that a transaction is composed of an indivisible unit of work. All operations performed in a transaction must either be committed or uncommitted. If an error occurs during transaction execution, the transaction is rolled back to the state when it was not committed.
online scale-out	Online scale-out means that data can be saved to the database and query services are not interrupted during redistribution in GaussDB(DWS).
dirty page	A page that has been modified and is not written to a permanent device.
incremental backup	Incremental backup stores all files changed since the last valid backup.
incremental synchronization	A data synchronization mechanism in the GaussDB(DWS) HA solution. Only data modified since the last synchronization is synchronized to the standby server.
Host	A node that receives data read and write operations in the GaussDB(DWS) HA system and works with all standby servers. At any time, only one node in the HA system is identified as the primary server.
thesaurus	Standardized words or phrases that express document themes and are used for indexing and retrieval.
dump file	A specific type of the trace file. A dump is typically a one-time output of diagnostic data in response to an event, whereas a trace tends to be continuous output of diagnostic data.
resource pool	Resource pools used for allocating resources in GaussDB(DWS). By binding a user to a resource pool, you can limit the priority of the jobs executed by the user and resources available to the jobs.
tenant	A database service user who runs services using allocated computing (CPU, memory, and I/O) and storage resources. Service level agreements (SLAs) are met through resource management and isolation.
minimum restoration point	A method used by GaussDB(DWS) to ensure data consistency. During startup, GaussDB(DWS) checks consistency between the latest WAL logs and the minimum restoration point. If the record location of the minimum restoration point is greater than that of the latest WAL logs, the database fails to start.