Data Warehouse Service

Developer Guide

Issue   06
Date   2023-04-12

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Contents

1 Welcome

1.1 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.

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 ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE PROVIDED HEREUNDER IS ON AN "AS-IS" BASIS, AND THE UNIVERSITY OF

CALIFORNIA HAS NO OBLIGATIONS TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

1.2 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.

If you want to...	Query Suggestions
Quickly get started with GaussDB(DWS).	Deploy a cluster, connect to the database, and perform some queries by following the instructions provided in **Getting Started**. 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) home page.
Learn how to design tables to achieve the excellent performance.	**Development and Design Proposal** 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 **Query Performance Optimization**. Successful performance optimization depends more on database administrators' experience and judgment than on instructions and explanation. However, **Query Performance Optimization** still tries to systematically illustrate the performance optimization methods for application development personnel and new GaussDB(DWS) database administrators.

If you want to...	Query Suggestions
Load data.	**Importing Data** describes how to import data to GaussDB(DWS). **Excellent Practices for Data Import** provides key points for quick data import.
Manage users, groups, and database security.	**Database Security Management** covers database security topics.
Monitor and optimize system performance.	**System Catalogs and System Views** describes the system catalogs where you can query the database status and monitor the query content and process. You can learn how to check the system running status and monitoring metrics on the GaussDB(DWS) console by referring to **Management Guide**.

1.3 Conventions

Example Conventions

Example	Description
dbadmin	Indicates the user operating and maintaining GaussDB(DWS) appointed when the cluster is created.
8000	Indicates the port number used by GaussDB(DWS) to monitor connection requests from the client.

SQL examples in this manual are developed based on the TPC-DS model. Before you execute the examples, install the TPC-DS benchmark by following the instructions on the official website **http://www.tpc.org/tpcds/**.

SQL Syntax Text Conventions

To better understand the syntax usage, you can refer to the SQL syntax text conventions described as follows:

Format	Description
Uppercase characters	Keywords must be in uppercase.
Lowercase characters	Indicates that parameters must be in lowercase.
[ ]	Indicates that the items in brackets [] are optional.

Format	Description
...	Indicates that preceding elements can appear repeatedly.
[x | y | ...]	Indicates that one item is selected from two or more options or no item is selected.
{ x | y | ... }	Indicates that one item is selected from two or more options.
[x | y | ...] [...]	Indicates that multiple parameters or no parameter can be selected. If multiple parameters are selected, separate them with spaces.
[x | y | ...] [,...]	Indicates that multiple parameters or no parameter can be selected. If multiple parameters are selected, separate them with commas (,).
{ x | y | ... } [...]	Indicates that at least one parameter can be selected. If multiple parameters are selected, separate them with spaces.
{ x | y | ... } [,...]	Indicates that at least one parameter can be selected. If multiple parameters are selected, separate them with commas (,).

1.4 Prerequisites

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

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

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

2 System Overview

2.1 Highly Reliable Transaction Processing

GaussDB(DWS) manages cluster transactions, the basis of HA and failovers. This ensures speedy fault recovery, guarantees the Atomicity, Consistency, Isolation, Durability (ACID) properties for transactions and after a recovery, and enables concurrent control.

Fault Rectification

GaussDB(DWS) provides an HA mechanism to reduce the service interruption time when a cluster is faulty. It protects key user programs to continuously provide external services, minimizing the impact of hardware, software, and human faults on services and ensuring service continuity.

- Hardware HA: Disk RAID, switch stacking, NIC bond, and uninterruptible power supply (UPS)
- Software HA: HA mechanism used for instances in the GaussDB(DWS) cluster, such as CNs, GTMs, and DNs)

Transaction Management

- Transaction blocks are supported. You can run **start transaction** to make the startup of a transaction block explicit.
- Single-statement transactions are supported. If you do not explicitly start a transaction, a single statement is processed as a transaction.
- Distributed transaction management and global transaction information management are supported. This includes gxid, snapshot, timestamp management, distributed transaction status management, and gxid overflow processing.
- Distributed transactions have ACID properties.
- Deadlocks are prevented in the distributed system. A transaction will be unlocked immediately after a deadlock (if any).

2.2 High Query Performance

The following GaussDB(DWS) features help achieve high query performance.

Fully Parallel Query

GaussDB(DWS) is an MPP system with the shared-nothing architecture. It consists of multiple independent logical nodes that do not share system resources, such as the CPU, memory, and storage units. In such a system architecture, service data is separately stored on numerous nodes. Data analysis tasks are executed in parallel on the nodes where data is stored. The massively parallel data processing significantly improves response speed.

In addition, GaussDB(DWS) improves data query performance by executing operators in parallel, executing commands in registers in parallel, and using LLVM to dynamically compile the logical conditions of redundancy prune.

Hybrid Row-Column Storage

GaussDB(DWS) supports both the row and column storage models. You can choose a row- or column-store table as needed.

The hybrid row-column storage engine achieves higher data compression ratio (column storage), index performance (column storage), and point update and point query (row storage) performance.

Data Compression in Column Storage

You can compress old, inactive data to free up space, reducing procurement and O&M costs.

In GaussDB(DWS), data can be compressed using the Delta Value Encoding, Dictionary, RLE, LZ4, and ZLIB algorithms. The system automatically selects a compression algorithm based on data characteristics. The average compression ratio is 7:1. Compressed data can be directly accessed and is transparent to services, greatly reducing the preparation time before accessing historical data.

2.3 Related Concepts

Database

A database manages data objects and is isolated from other databases. While creating an object, you can specify a tablespace for it. If you do not specify it, the object will be saved to the **PG_DEFAULT** space by default. Objects managed by a database can be distributed to multiple tablespaces.

Instance

In GaussDB(DWS), instances are a group of database processes running in the memory. An instance can manage one or more databases that form a cluster. A cluster is an area in the storage disk. This area is initialized during installation and

composed of a directory. The directory, called data directory, stores all data and is created by **initdb**. Theoretically, one server can start multiple instances on different ports, but GaussDB(DWS) manages only one instance at a time. The start and stop of an instance rely on the specific data directory. For compatibility purposes, the concept of instance name may be introduced.

Tablespaces

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

schema

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

User and Role

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

Transaction Management

In GaussDB(DWS), transactions are managed by multi-version concurrency control (MVCC) and two-phase locking (2PL). It enables smooth data reads and writes. In GaussDB(DWS), MVCC saves historical version data together with the current tuple version. GaussDB(DWS) uses the VACUUM process instead of rollback segments to routinely delete historical version data. Unless in performance optimization, you do not need to pay attention to the VACUUM process. Transactions are automatically submitted in GaussDB(DWS).

3 Defining Database Objects

3.1 Creating and Managing 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** command to create a database.

CREATE DATABASE *mydatabase*;

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

📖 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).

Viewing Databases

To view databases, perform the following steps:

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

Modifying a Database

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

- Run the following command to set the default search path for the database:
  ALTER DATABASE *mydatabase* **SET** *search_path* **TO** *pa_catalog,public*;
- Run the following command to rename the database:
  ALTER DATABASE *mydatabase* **RENAME TO** *newdatabase*;

Deleting a Database

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

Run the following command to delete a database:
DROP DATABASE *newdatabase*;

3.2 Creating and Managing 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.

The Default Schema Public

Each database has a schema named **Public**. If you do not create any schema, the object will be created in the schema named public. All database roles (users) have the CREATE and USAGE permissions in the public schema. When creating a schema, you need to grant the access permission to users.

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;
  ```

- 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 Creating and Managing 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, 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 Table Storage Mode**.

- Partition table information. For details, see **Defining Table Partitions** .

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:

1. The primary key constraint and unique constraint in the table must contain a distribution column.

2. 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.

3. Only the **NULL**, **NOT NULL**, and **DEFAULT** constant values can be used as column-store table column constraints.

- CHECK constraint

  A CHECK constraint allows you to specify that values in a specific column must satisfy a Boolean (true) expression. For example, the product price 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. For example:

  ```
  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 TABLE products
  (
      product_no integer UNIQUE,
      name text,
      price numeric
  )DISTRIBUTE BY HASH(product_no);
  ```

- Primary key

  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.

For example:

```
CREATE TABLE products
(
    product_no integer PRIMARY KEY,
    name text,
    price numeric
)DISTRIBUTE BY HASH(product_no);
```

- Partial cluster key

  Partial cluster key can minimize or maximize sparse indexes to quickly filter base tables. Partial cluster keys can specify multiple columns, but you are advised to specify no more than two columns. For example:

```
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.

Policy	Description	Scenario	Advantages/Disadvantages
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.
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:

  a. **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.

  b. **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.

  c. **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.**

d. 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

You can delete outdated data from a table by row.

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

- For example, to delete all the rows whose **c_customer_sk** column is **3869** from table **customer_t1**, run the following statement:
  DELETE FROM *customer_t1* **WHERE** *c_customer_sk = 3869*;

- To delete all rows from the table, run either of the following statements:
  DELETE FROM *customer_t1*;
  TRUNCATE TABLE *customer_t1*;

📖 **NOTE**

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

● Delete the created table.
DROP TABLE *customer_t1*;

3.4 Selecting a Table Storage Mode

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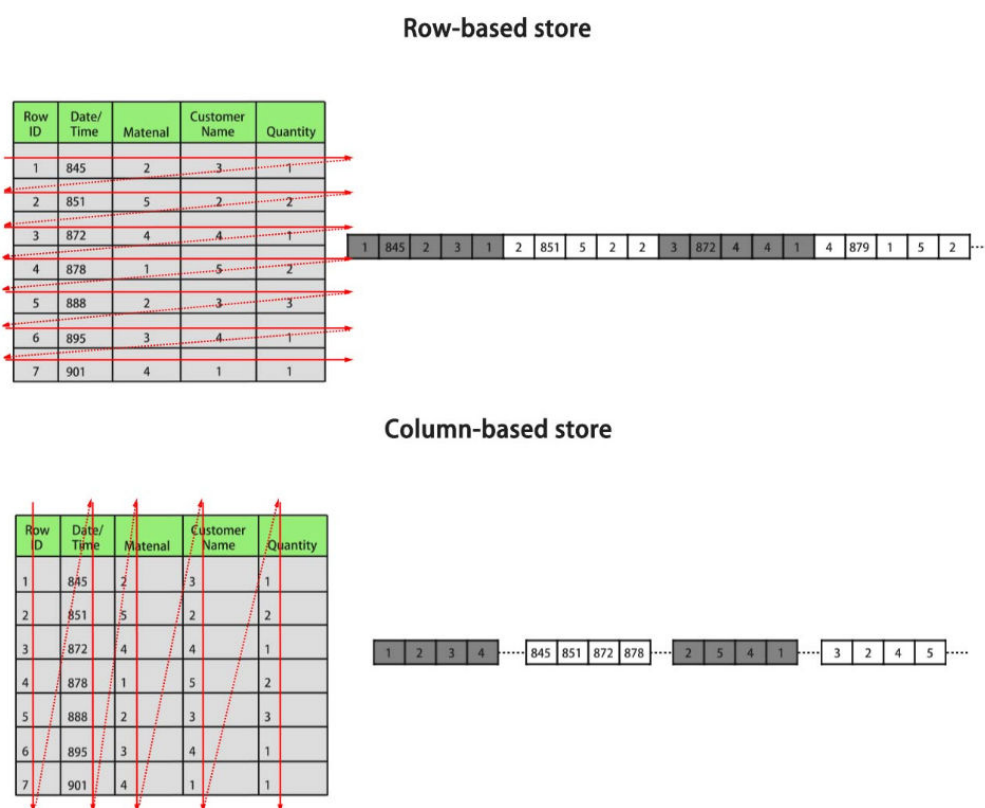
In the preceding figure, the upper left part is a row-store table, and the upper right part shows how the row-store table is stored on a disk; the lower left part is a column-store table, and the lower right part shows how the column-store table is stored on a disk.

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.

Storage Mode	Benefit	Drawback	Application Scenarios
Row storage	All the columns of a record are stored in the same partition. Data can be easily inserted and updated.	All the columns of a record are read after the **SELECT** statement is executed even if only certain columns are required.	- Query most columns in a table that has a relatively small number of columns. - Point query (simple index–based query that returns only a few records). - Scenarios requiring frequent addition, deletion, and modification.
Column storage	- Only necessary columns in a query are read. - Projections are efficient. - Any column can serve as an index.	- The selected columns need to be reconstructed after the **SELECT** statement is executed. - Data cannot be easily inserted or updated.	- Query a few columns in a table that contains a large number of columns. - Statistical analysis queries (requiring a large number of association and grouping operations) - Ad hoc queries (using uncertain query conditions and unable to utilize indexes to scan row-store tables)

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);
```

3.5 Defining Table Partitions

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 source, 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)
);
```

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 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**.

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 3-1**.

📖 **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 3-1 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 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 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.

- Run the CREATE SEQUENCE statement to create a sequence. Set the initial value of the **nextval**('*sequence_name*') function to the default value of a column.

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. The GTM is the only node that can generate sequence values and probably becomes the performance bottleneck. Therefore, you are not advised to use sequences when sequence values need to be generated frequently (for example, using BulkLoad to import data). 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 effects on nextval in the current session in any cases and take effects 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 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 Scheduled Tasks

When a customer executes some time-consuming tasks during the day time, (for example, statistics summary task or other database synchronization tasks), the service performance will be influenced. So customers execute tasks on database during night time, increasing the workload. The scheduled task function of the database is compatible with the Oracle database scheduled task function that customers can create scheduled tasks. When the scheduled task time arrives, the task will be triggered. Therefore, the workload of OM has been reduced.

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.

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.

```
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.

  -- Specify the task of modifying **Next_date** of **Job1** will be executed in one hour.
  ```
  call dbms_job.next_date(1, sysdate+1.0/24);
  next_date
  -----------

  (1 row)
  ```

- Modify the **Interval** parameter information of a task.

  -- Set **Interval** of **Job1** to **1**.
  ```
  call dbms_job.interval(1,'sysdate + 1.0/24');
  interval
  ----------

  (1 row)
  ```

- Modify the **What** parameter information of a **JOB**.

  -- Change **What** to the SQL statement **insert into public.test values(333, sysdate+5);** for **Job1**.
  ```
  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 a System Catalog

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.

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

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;
 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)
```

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 Data migration

4.1 Data Migration to GaussDB(DWS)

GaussDB(DWS) provides flexible methods for importing data. You can import data from different sources to GaussDB(DWS). The features of each method are listed in **Table 4-1**. You can select a method as required. You are advised to use GaussDB(DWS) with Data Replication Service (DRS), Cloud Data Migration (CDM), Data Ingestion Service (DIS), and Data Lake Factory (DLF). DRS is used for real-time data synchronization, CDM is used for batch data migration, DIS is used for stream data ingestion, and DLF orchestrates and schedules the entire ETL process and provides a visualized development environment.

Figure 4-1 Data migration
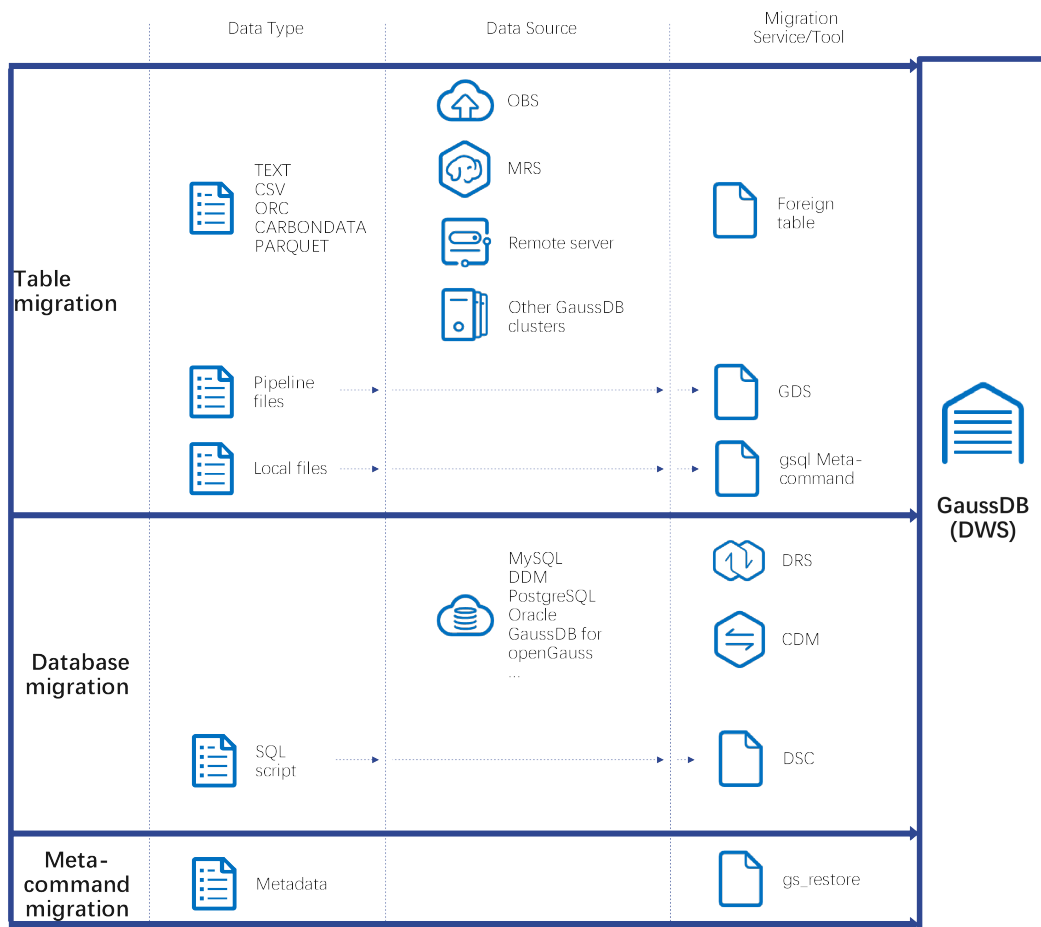
> **NOTE**
>
> ● DRS, CDM, OBS, and MRS are cloud services.
> ● GDS, DSC, and **gs_restore**, and **gs_dump** are internal tools.

Table 4-1 Import methods

Import Method	Data Source	Description	Advantage
Importing Data from OBS in Parallel	OBS	You can import data in TXT, CSV, ORC, or CarbonData format from OBS to GaussDB(DWS) for query, and can remotely read data from OBS. It is recommended for GaussDB(DWS).	This method features high performance and flexible scale-out.

Import Method	Data Source	Description	Advantage
Using GDS to Import Data from a Remote Server	Servers (remote servers)	Use the GDS tool provided by GaussDB(DWS) to import data from the remote server to GaussDB(DWS) in parallel. Multiple DNs are used for the import. This method is efficient and applicable to importing a large amount of data to the database.	
Importing Data from MRS to a Cluster	MRS (HDFS)	Configure a GaussDB(DWS) cluster to connect to an MRS cluster. In GaussDB(DWS), read data from the HDFS of MRS.	This method features high performance and flexible scale-out.
Importing Data from One GaussDB(DWS) Cluster to Another	-	Two GaussDB(DWS) clusters can access data of each other. You can use foreign tables to access and import data across GaussDB(DWS) clusters.	This method is applicable to data synchronization between multiple GaussDB(DWS) clusters.
GDS-based Cross-Cluster Interconnection	-	GDS is used for data transit to implement data synchronization between multiple clusters.	This method is applicable to data synchronization between multiple GaussDB(DWS) clusters.
Using a gsql Meta-Command to Import Data	Local files	Unlike the SQL **COPY** statement, the **\copy** command can read data from or write data into only local files on a **gsql** client.	This method is easy to operate and suitable for importing a small amount of data to the database.

Import Method	Data Source	Description	Advantage
Running the COPY FROM STDIN Statement to Import Data	Other files or databases	When you use Java to develop applications, the CopyManager interface of the JDBC driver is invoked to write data from files or other databases to GaussDB(DWS).	Data is directly written from other databases to GaussDB(DWS). Service data does not need to be stored in files.
Using DRS to Import Data to GaussDB(DWS)	• MySQL • DDM • PostgreSQL (OBT) • Oracle (OBT) • GaussDB Distributed (OBT)	The real-time data synchronization of the DRS allows you to copy data from a data source to GaussDB(DWS) to implement real-time data flow of key services. This method focuses on the synchronous import of tables and data.	This method supports data import from abundant data sources and is easy-to-operate.
Using CDM to Migrate Data to GaussDB(DWS)	Databases, NoSQL, file systems, and big data platforms	CDM can migrate various types of data in batches between homogeneous and heterogeneous data sources. CDM migrates data to GaussDB(DWS) using the copy method or the GDS parallel import method.	This method supports data import from abundant data sources and is easy-to-operate.
Using DSC to Migrate SQL Scripts	Databases, NoSQL, file systems, and big data platforms	For details, see the documents of the third-party ETL tool. GaussDB(DWS) provides the DSC tool to migrate Teradata/Oracle scripts to GaussDB(DWS).	Provides abundant data sources and powerful data conversion capabilities through OBS.

Import Method	Data Source	Description	Advantage
Using gs_dump and gs_dumpall to Export Metadata	- Plain text - Custom - Directory .tar	gs_dump exports a single database or its objects. gs_dumpall exports all databases or global objects in a cluster. To migrate database information, you can use a tool to import the exported metadata to a target database.	This method is applicable to metadata migration.
Using gs_restore to Import Data	SQL, TMP, and TAR file formats	During database migration, you can use the gs_restore tool to import the file exported using the gs_dump tool to a GaussDB(DWS) cluster. In this way, metadata, such as table definitions and database object definitions, is imported. The following definitions need to be imported: - All database objects - A single database object - A single schema - A single table	

4.2 Importing Data

4.2.1 Importing Data from OBS in Parallel

4.2.1.1 About Parallel Data Import from OBS

The object storage service (OBS) is an object-based cloud storage service, featuring data storage of high security, proven reliability, and cost-effectiveness. OBS provides large storage capacity for you to store files of any type.

GaussDB(DWS), a data warehouse service, uses OBS as a platform for converting cluster data and external data, satisfying the requirements for secure, reliable, and cost-effective storage.

You can import data in TXT, CSV, ORC, CARBONDATA, or JSON format from OBS to GaussDB(DWS) for query, and can remotely read data from OBS. You are advised to import frequently accessed hot data to GaussDB(DWS) to facilitate queries and store cold data to OBS for remote read to reduce cost.

Currently, data can be imported using either of the following methods:

- Method 1: You do not need to create a server. Use the default server to create a foreign table. Data in TXT or CSV format is supported. For details, see **Importing CSV/TXT Data from the OBS**.

- Method 2: You need to create a server and use the server to create a foreign table. Data in ORC, CarbonData, TXT, CSV, or JSON format is supported. For details, see **Importing ORC or CarbonData Data from OBS**.

NOTICE

- Ensure that no Chinese characters are contained in paths used for importing data to or exporting data from OBS.

- Data cannot be imported to or exported from OBS across regions. Ensure that OBS and the DWS cluster are in the same region.

Overview

During data migration and Extract-Transform-Load (ETL), a massive volume of data needs to be imported to GaussDB(DWS) in parallel. The common import mode is time-consuming. When you import data in parallel using OBS foreign tables, source data files to be imported are identified based on the import URL and data formats specified in the tables. Data is imported in parallel through DNs to GaussDB(DWS), which improves the overall import performance.

Advantages:

- The CN only plans and delivers data import tasks, and the DNs execute these tasks. This reduces CN resource usage, enabling the CN to process external requests.

- In this way, the computing capabilities and bandwidths of all the DNs are fully leveraged to import data.

- You can preprocess data before the import.

- Fault tolerance can be configured for data format errors during the data import. You can locate incorrect data based on displayed error information after the data is imported.

Disadvantage:

You need to create OBS foreign tables and store to-be-imported data on OBS.

Application Scenario:

A large volume of local data is imported concurrently on many DNs.

Related Concepts

- **Source data file**: a TEXT, CSV, ORC, CARBONDATA, or JSON file that stores data to be imported in parallel.

- **OBS**: a cloud storage service used to store unstructured data, such as documents, images, and videos. Data is imported in parallel from the OBS server to GaussDB(DWS).

- **Bucket**: a container storing objects on OBS.
  - Object storage is a flat storage mode. Layered file system structures are not needed because all objects in buckets are at the same logical layer.
  - In OBS, each bucket name must be unique and cannot be changed. A default access control list (ACL) is created with a bucket. Each item in the ACL contains permissions granted to certain users, such as **READ**, **WRITE**, and **FULL_CONTROL**. Only authorized users can perform bucket operations, such as creating, deleting, viewing, and setting ACLs for buckets.
  - A user can create a maximum of 100 buckets. The total data size and the number of objects and files in each bucket are not limited.
- **Object**: a basic data storage unit in OBS. Data uploaded by users is stored in OBS buckets as objects. Object attributes include **Key**, **Metadata**, and **Data**.

  Generally, objects are managed as files. However, OBS has no file system–related concepts, such as files and folders. To let users easily manage data, OBS allows them to simulate folders. Users can add a slash (/) in the object name, for example, **tpcds1000/stock.csv**. In this name, **tpcds1000** is regarded as the folder name and **stock.csv** the file name. The value of **key** (object name) is still **tpcds1000/stock.csv**, and the content of the object is the content of the **stock.csv** file.
- **Key**: name of an object. It is a UTF-8 character sequence containing 1 to 1024 characters. A key value must be unique in a bucket. Users can name the objects they stored or obtained as *Bucket name+Object name*.
- **Metadata**: object metadata, which contains information about the object. There are system metadata and user metadata. The metadata is uploaded to OBS as key-value pairs together with HTTP headers.
  - System metadata is generated by OBS and used for processing object data. System metadata includes **Date**, **Content-length**, **last-modify**, and **Content-MD5**.
  - User metadata contains object descriptions specified by users for uploading objects.
- **Data**: object content. OBS does not sense the content and regards it as stateless binary data.
- **Ordinary table**: A database table that stores data imported to data files in parallel. Ordinary tables are classified into row-store tables and column-store tables.
- **Foreign table**: A foreign table is used to identify data in a source data file. The foreign table stores information, such as the location, format, encoding, and inter-data delimiter of a source data file.

How Data Is Imported

Figure 4-2 shows how data is imported from OBS. The CN plans and delivers data import tasks. It delivers tasks to each DN by file.

The delivery method is as follows:

In **Figure 4-2**, there are four DNs (DN0 to DN3) and OBS stores six files numbered from t1.data.0 to t1.data.5. The files are delivered as follows:

t1.data.0 -> DN0

t1.data.1 -> DN1

t1.data.2 -> DN2

t1.data.3 -> DN3

t1.data.4 -> DN0

t1.data.5 -> DN1

Two files are delivered to DN0 and DN1, respectively. One file is delivered to each of the other DNs.

The import performance is the best when one OBS file is delivered to each DN and all the files have the same size. To improve the performance of loading data from OBS, split the data file into multiple files as evenly as possible before storing it to OBS. The recommended number of split files is an integer multiple of the DN quantity.

Figure 4-2 Parallel data import using OBS foreign tables
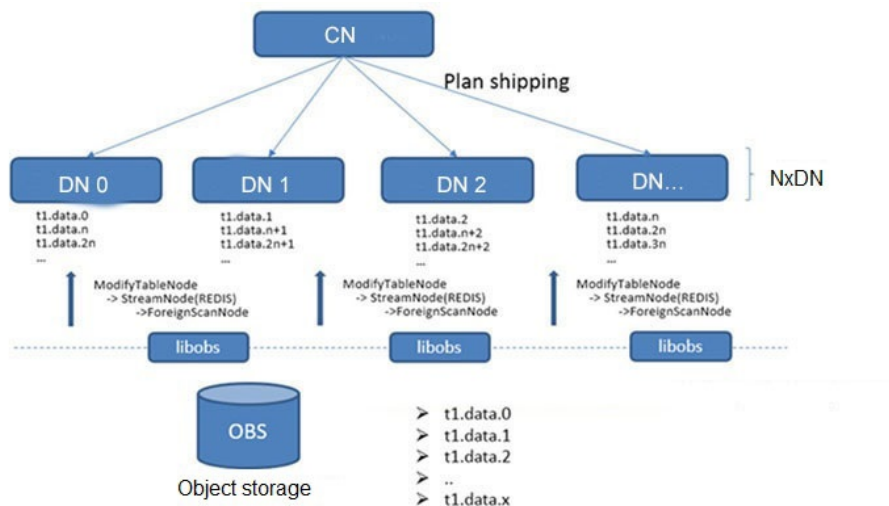
Import Flowchart

Figure 4-3 Parallel import procedure
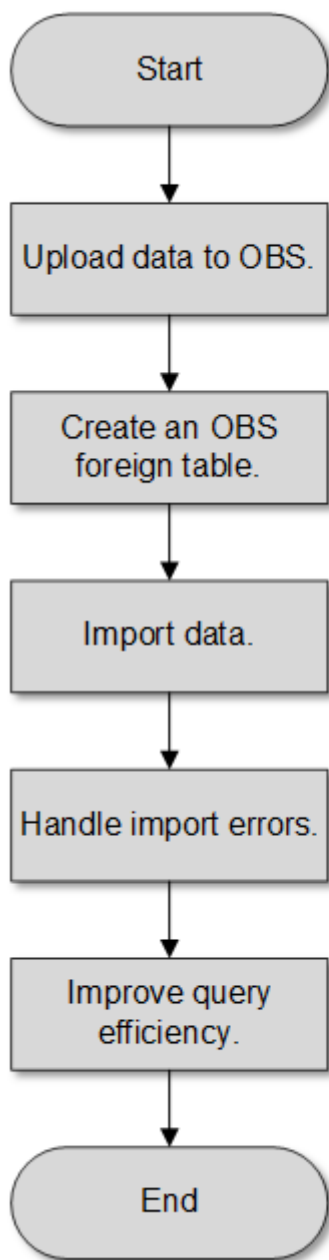
Table 4-2 Procedure description

Procedure	Description	Subtask
Upload data to OBS.	Plan the storage path on the OBS server and upload data files. For details, see **Uploading Data to OBS**.	-

Procedure	Description	Subtask
Create an OBS foreign table.	Create a foreign table to identify source data files on the OBS server. The OBS foreign table stores data source information, such as its bucket name, object name, file format, storage location, encoding format, and delimiter. For details, see **Creating an OBS Foreign Table**.	-
Import data.	After creating the foreign table, run the **INSERT** statement to efficiently import data to the target tables. For details, see **Importing Data**.	-
Handle the table with import errors.	If errors occur during data import, handle them based on the displayed error information described in **Handling Import Errors** to ensure data integrity.	-
Improve query efficiency.	After data is imported, run the **ANALYZE** statement to generate table statistics. The **ANALYZE** statement stores the statistics in the **PG_STATISTIC** system catalog. When you run the plan generator, the statistics help you generate an efficient query execution plan.	-

4.2.1.2 Importing CSV/TXT Data from the OBS

4.2.1.2.1 Creating Access Keys (AK and SK)

In this example, OBS data is imported to GaussDB(DWS) databases. When users who have registered with the cloud platform access OBS using clients, call APIs, or SDKs, access keys (AK/SK) are required for user authentication. Therefore, if you want to connect to the GaussDB(DWS) database through a client or a JDBC/ODBC application to access OBS, obtain the access keys (AK and SK) first.

- Access Key ID (AK): indicates the ID of the access key, which is a unique identifier used in conjunction with a Secret Access Key to sign requests cryptographically.
- Secret Access Key (SK): indicates the key used with its associated AK to cryptographically sign requests and identify request senders to prevent requests from being modified.

Creating Access Keys (AK and SK)

Before creating an AK/SK pair, ensure that your account (used to log in to the management console) has passed real-name authentication.

To create an AK/SK pair on the management console, perform the following steps:

Step 1 Log in to the GaussDB(DWS) management console.

Step 2 Click the username in the upper right corner and choose **My Credentials** from the drop-down list.

Step 3 In the navigation tree on the left, click **Access Keys**.

If an access key already exists in the access key list, you can directly use it. However, you can view only **Access Key ID** in the access key list. You can download the key file containing the AK and SK only when adding an access key. If you do not have the key file, click **Create Access Key** to create one.

☐ NOTE

- Each user can create a maximum of two valid access keys. If there are already two access keys, delete them and create one. To delete an access key, you need to enter the current login password and email address or SMS verification code. Deletion is successful only after the verification is passed.
- To ensure account security, change your access keys periodically and keep them secure.

Step 4 Click **Add Access Key**.

Step 5 In the displayed **Add Access Key** dialog box, enter the password and its verification code and click **OK**.

☐ NOTE

- If you have not bound an email address or a mobile number, enter only the login password.
- If you have bound an email address and a mobile phone number, you can use either of them for verification.

Step 6 In the displayed **Download Access Key** dialog box, click **OK** to save the access keys to your browser's default download path.

☐ NOTE

- Keep the access keys secure to prevent them from being leaked.
- If you click **Cancel** in the dialog box, the access keys will not be downloaded, and you cannot download them later. In this case, re-create access keys.

Step 7 Open the downloaded **credentials.csv** file to obtain the access keys (AK and SK).

----End

Precautions

If you find that your AK/SK pair is abnormally used (for example, the AK/SK pair is lost or leaked) or will be no longer used, delete your AK/SK pair in the access key list or contact the administrator to reset your AK/SK pair.

When deleting the access keys, you need to enter the login password and either an email or mobile verification code.

> **NOTE**
>
> Deleted AK/SK pairs cannot be restored.

4.2.1.2.2 Uploading Data to OBS

Scenarios

Before importing data from OBS to a cluster, prepare source data files and upload these files to OBS. If the data files have been stored on OBS, you only need to complete **Step 2** to **Step 3** in **Uploading Data to OBS**.

Preparing Data Files

Prepare source data files to be uploaded to OBS. GaussDB(DWS) supports only source data files in CSV, TXT, ORC, or CarbonData format.

If user data cannot be saved in CSV format, store the data as any text file.

> **NOTE**
>
> According to **How Data Is Imported**, when the source data file contains a large volume of data, evenly split the file into multiple files before storing it to OBS. The import performance is better when the number of files is an integer multiple of the DN quantity.

Assume that you have stored the following three CSV files in OBS:

- Data file **product_info.0**

  The file contains the following data:

  ```
  100,XHDK-A-1293-#fJ3,2017-09-01,A,2017 Autumn New Shirt Women,red,M,
  328,2017-09-04,715,good!
  205,KDKE-B-9947-#kL5,2017-09-01,A,2017 Autumn New Knitwear Women,pink,L,
  584,2017-09-05,406,very good!
  300,JODL-X-1937-#pV7,2017-09-01,A,2017 autumn new T-shirt men,red,XL,1245,2017-09-03,502,Bad.
  310,QQPX-R-3956-#aD8,2017-09-02,B,2017 autumn new jacket women,red,L,411,2017-09-05,436,It's
  really super nice.
  150,ABEF-C-1820-#mC6,2017-09-03,B,2017 Autumn New Jeans Women,blue,M,
  1223,2017-09-06,1200,The seller's packaging is exquisite.
  ```

- Data file **product_info.1**

  The file contains the following data:

  ```
  200,BCQP-E-2365-#qE4,2017-09-04,B,2017 autumn new casual pants men,black,L,
  997,2017-09-10,301,The clothes are of good quality.
  250,EABE-D-1476-#oB1,2017-09-10,A,2017 autumn new dress women,black,S,
  841,2017-09-15,299,Follow the store for a long time.
  108,CDXK-F-1527-#pL2,2017-09-11,A,2017 autumn new dress women,red,M,85,2017-09-14,22,It's
  really amazing to buy.
  450,MMCE-H-4728-#nP9,2017-09-11,A,2017 autumn new jacket women,white,M,
  114,2017-09-14,22,Open the package and the clothes have no odor.
  260,OCDA-G-2817-#bD3,2017-09-12,B,2017 autumn new woolen coat women,red,L,
  2004,2017-09-15,826,Very favorite clothes.
  ```

- Data file **product_info.2**

  The file contains the following data:

  ```
  980,"ZKDS-J",2017-09-13,"B","2017 Women's Cotton Clothing","red","M",112,,,
  98,"FKQB-I",2017-09-15,"B","2017 new shoes men","red","M",4345,2017-09-18,5473
  50,"DMQY-K",2017-09-21,"A","2017 pants men","red","37",28,2017-09-25,58,"good","good","good"
  80,"GKLW-l",2017-09-22,"A","2017 Jeans Men","red","39",58,2017-09-25,72,"Very comfortable."
  30,"HWEC-L",2017-09-23,"A","2017 shoes women","red","M",403,2017-09-26,607,"good!"
  40,"IQPD-M",2017-09-24,"B","2017 new pants Women","red","M",35,2017-09-27,52,"very good."
  50,"LPEC-N",2017-09-25,"B","2017 dress Women","red","M",29,2017-09-28,47,"not good at all."
  60,"NQAB-O",2017-09-26,"B","2017 jacket women","red","S",69,2017-09-29,70,"It's beautiful."
  70,"HWNB-P",2017-09-27,"B","2017 jacket women","red","L",30,2017-09-30,55,"I like it so much"
  80,"JKHU-Q",2017-09-29,"C","2017 T-shirt","red","M",90,2017-10-02,82,"very good."
  ```

Uploading Data to OBS

Step 1 Upload data to OBS.

Store the source data files to be imported in the OBS bucket in advance.

1. Log in to the OBS management console.

   Click **Service List** and choose **Object Storage Service** to open the OBS management console.

2. Create a bucket.

   For details about how to create an OBS bucket, see **Creating a Bucket** in the *Object Storage Service Console Operation Guide*.

   For example, create two buckets named **mybucket** and **mybucket02**.

3. Create a folder.

   For details, see **Creating a Folder** in *Object Storage Service Console Operation Guide*.

   For example:

   – Create a folder named **input_data** in the **mybucket** OBS bucket.

   – Create a folder named **input_data** in the **mybucket02** OBS bucket.

4. Upload the files.

   For details, see section **Uploading a File** in *Object Storage Service Console Operation Guide*.

   For example:

   – Upload the following data files to the **input_data** folder in the **mybucket** OBS bucket:
     ```
     product_info.0
     product_info.1
     ```

   – Upload the following data file to the **input_data** folder in the **mybucket02** OBS bucket:
     ```
     product_info.2
     ```

Step 2 Obtain the OBS path for storing source data files.

After the source data files are uploaded to an OBS bucket, a globally unique access path is generated. The OBS path of the source data files is the value of the **location** parameter used for creating a foreign table.

The OBS path in the **location** parameter is in the format of **obs://***bucket_name*/*file_path*/

For example, the OBS paths are as follows:

```
obs://mybucket/input_data/product_info.0
obs://mybucket/input_data/product_info.1
obs://mybucket02/input_data/product_info.2
```

Step 3 Grant the OBS bucket read permission for the user who will import data.

When importing data from OBS to a cluster, the user must have the read permission for the OBS buckets where the source data files are located. You can configure the ACL for the OBS buckets to grant the read permission to a specific user.

For details, see **Configuring a Bucket ACL** in *Object Storage Service Console Operation Guide*.

----End

4.2.1.2.3 Creating an OBS Foreign Table

Procedure

Step 1 Set **location** of the foreign table based on the path planned in **Uploading Data to OBS**.

Step 2 Obtain the access keys (AK and SK) to access OBS. To obtain access keys, log in to the management console, move the cursor over the username in the upper right corner, and click **My Credentials**. Then choose **Access Keys** in the navigation tree on the left.

Step 3 Set data format parameters for the foreign table based on the formats of data to be imported. You need to collect the following source data information:

- **format**: format of the source data file in the foreign table. OBS foreign tables support CSV and TEXT formats. The default value is **TEXT**.

- **header**: Whether the data file contains a table header. Only CSV files can have headers.

- **delimiter**: Delimiter specified to separate data fields in a file. If no delimiter is specified, the default one will be used.

- For more parameters used for foreign tables, see data format parameters.

Step 4 Plan the error tolerance of parallel import to specify how errors are handled during the import.

- **fill_missing_fields**: When the last column in a row of the source data file is empty, this parameter specifies whether to report an error or set this field in the row to **NULL**.

- **ignore_extra_data**: When the number of columns in the source data file is greater than that specified in the foreign table, this parameter specifies whether to report an error or ignore the extra columns.

- **per node reject limit**: This parameter specifies the number of data format errors allowed on each DN. If the number of errors recorded in the error table on a DN exceeds the specified value, the import fails and an error message will be reported. This parameter is optional.

- **compatible_illegal_chars**: When an illegal character is encountered, this parameter specifies whether to import an error, or convert it and proceed with the import.

  The following describes the rules for converting an invalid character:

- **\0** is converted to a space.

- Other invalid characters are converted to question marks (?).

- If **NULL**, **DELIMITER**, **QUOTE**, or **ESCAPE** is also set to a space or question mark, an error message such as "illegal chars conversion may confuse COPY escape 0x20" is displayed, prompting you to modify parameter settings that may cause import errors.

- **error_table_name**: This parameter specifies the name of the table that records data format errors. After the parallel import, you can query this table for error details.

- For details about the parameters, see error tolerance parameters.

Step 5 Create an OBS table based on the parameter settings in the preceding steps. For details about how to create a foreign table, see CREATE FOREIGN TABLE (for GDS Import and Export).

----End

Example

Create a foreign table in the GaussDB(DWS) database. Parameters are described as follows:

- **Data format parameter access keys (AK and SK)**

  - Set **access_key** to the AK you have obtained.

  - Set **secret_access_key** to the SK you have obtained.

  📖 **NOTE**

  The values of **access_key** and **secret_access_key** are examples only.

- **Set data format parameters.**

  - Set **format** to **CSV**.

  - Set **encoding** to **UTF-8**.

  - Configure **encrypt**. Its default value is **off**.

  - Set **delimiter** to **,**.

  - Retain the default value (double quotation marks) of **quote**.

  - Set **null** (null value in a source data file) to a null string without quotation marks.

  - Set **header** (whether the exported data file contains the header row) to the default value **false**. If the first row of the data file is not a header, retain the default value.

    📖 **NOTE**

    When exporting data from OBS, this parameter cannot be set to **true**. Use the default value **false**.

- **Set fault-tolerant parameters for data import.**

  - Set **PER NODE REJECT LIMIT** to **'unlimited'**, indicating that all data format errors detected during data import are allowed.

  - Set **LOG INTO** to **product_info_err**, indicating that the data format errors detected during data import are recorded in the **product_info_err** table.

- – If **fill_missing_fields** is set to **true** and the last column of a data row in a source data file is lost, the column is replaced with **NULL** and no error message will be generated.
- – If **ignore_extra_data** is set to **true** and the number of columns of the source data file is greater than that defined for the foreign table, the extra columns at the end of the row are ignored and no error message will be generated.

Based on the preceding settings, the foreign table is created using the following statements:

```
DROP FOREIGN TABLE product_info_ext;

CREATE FOREIGN TABLE product_info_ext
(
    product_price              integer       not null,
    product_id                 char(30)      not null,
    product_time               date        ,
    product_level              char(10)      ,
    product_name                varchar(200)  ,
    product_type1               varchar(20)   ,
    product_type2               char(10)      ,
    product_monthly_sales_cnt   integer       ,
    product_comment_time        date        ,
    product_comment_num         integer       ,
    product_comment_content     varchar(200)
)
SERVER gsmpp_server
OPTIONS(

LOCATION 'obs://mybucket/input_data/product_info | obs://mybucket02/input_data/product_info',
FORMAT 'CSV' ,
DELIMITER ',',
encoding 'utf8',
header 'false',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced',
fill_missing_fields 'true',
ignore_extra_data 'true'
)
READ ONLY
LOG INTO product_info_err
PER NODE REJECT LIMIT 'unlimited';
```

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

```
CREATE FOREIGN TABLE
```

4.2.1.2.4 Importing Data

Context

Before importing data, you are advised to optimize your design and deployment based on the following excellent practices, helping maximize system resource utilization and improving data import performance.

- In most cases, OBS data import performance is limited by concurrent network access rate. Therefore, you are advised to deploy multiple buckets on the OBS server to import data in parallel from buckets, better utilizing DN data transfer.
- Similar to the single table import, ensure that the I/O performance is greater than the maximum network throughput in the concurrent import.

- Set GUC parameters **raise_errors_if_no_files**, **partition_mem_batch**, and **partition_max_cache_size**. When importing data, specify whether to distinguish between the following two cases: no records exist in the data file or the data file does not exist. You also need to specify the number of caches and the size of data buffers.

- If a table has an index, the index information is incrementally updated during the import, affecting data import performance. You are advised to delete the index from the target table before the import. You can create index again after the import is complete.

Procedure

Step 1 Create a table in the GaussDB(DWS) database to store the data imported from the OBS. For details about the syntax, see CREATE TABLE.

The structure of the table must be consistent with that of the fields in the source data file. That is, the number of fields and field types must be the same. In addition, the structure of the target table must be the same as that of the foreign table. The field names can be different.

Step 2 (Optional) If the target table has an index, the index information is incrementally updated during the import, affecting data import performance. You are advised to delete the index from the target table before the import. You can create index again after the import is complete.

Step 3 Import data.

INSERT INTO *[Target table name]* **SELECT * FROM** *[Foreign table name]*

- If information similar to the following is displayed, the data has been imported. Query the error information table to check whether any data format errors occurred. For details, see **Handling Import Errors**.
  INSERT 0 20

- If data fails to be loaded, rectify the problem by following the instructions provided in **Handling Import Errors** and try again.

----End

Example

For example, create a table named *product_info*.

```
DROP TABLE IF EXISTS product_info;
CREATE TABLE product_info
(
    product_price                integer        not null,
    product_id                   char(30)       not null,
    product_time                 date         ,
    product_level                char(10)        ,
    product_name                 varchar(200)   ,
    product_type1                varchar(20)    ,
    product_type2                char(10)        ,
    product_monthly_sales_cnt    integer         ,
    product_comment_time         date           ,
    product_comment_num          integer          ,
    product_comment_content      varchar(200)
)
with (
orientation = column,
compression=middle
```

```
)
DISTRIBUTE BY HASH (product_id);
```

Run the following statement to import data from the *product_info_***ext** foreign table to the *product_info* table:

```
INSERT INTO product_info SELECT * FROM product_info_ext;
```

4.2.1.2.5 Handling Import Errors

Scenarios

Handle errors that occurred during data import.

Querying Error Information

Errors that occur when data is imported are divided into data format errors and non-data format errors.

- Data format error

  When creating a foreign table, specify **LOG INTO** *error_table_name*. Data format errors occurring during the data import will be written into the specified table. You can run the following SQL statement to query error details:

  ```
  SELECT * FROM error_table_name;
  ```

  Table 4-3 lists the columns of the **error_table_name** table.

  Table 4-3 Columns of the **error_table_name** table

Column	Type	Description
nodeid	integer	ID of the node where an error is reported
begintime	timestamp with time zone	Time when a data format error is reported
filename	character varying	Name of the source data file where an error about data format occurs
rownum	bigint	Number of the row where an error occurs in a source data file
rawrecord	text	Raw record of the data format error in the source data file
detail	text	Error details

- Non-data format error

  A non-data format error leads to the failure of an entire data import task. You can locate and troubleshoot a non-data format error based on the error message displayed during data import.

Handling data import errors

Troubleshoot data import errors based on obtained error information and the description in the following table.

Table 4-4 Handling data import errors

Error Information	Cause	Solution

missing data for column "r_reason_desc"	1. The number of columns in the source data file is less than that in the foreign table.
	
	2. In a TEXT format source data file, an escape character (for example, \) leads to delimiter or quote mislocation. Example: The target table contains three columns as shown in the following command output. The escape character (\) converts the delimiter (|) into the value of the second column, causing loss of the value of the third column.
	BE|Belgium\\\|1
		1. If an error is reported due to missing columns, perform the following operations:
		
		  ● Add the **r_reason_desc** column to the source data file.
		
		  ● When creating a foreign table, set the parameter **fill_missing_fields** to **on**. In this way, if the last column of a row in the source data file is missing, it is set to **NULL** and no error will be reported.
		
		2. Check whether the row where an error occurred contains the escape character (\). If the row contains such a character, you are advised to set the parameter **noescaping** to **true** when creating a foreign table, indicating that the escape character (\) and the characters following it are not escaped.

Error Information	Cause	Solution
extra data after last expected column	The number of columns in the source data file is greater than that in the foreign table.	• Delete the unnecessary columns from the source data file. • When creating a foreign table, set the parameter **ignore_extra_data** to **on**. In this way, if the number of columns in a source data file is greater than that in the foreign table, the extra columns at the end of rows will not be imported.
invalid input syntax for type numeric: "a"	The data type is incorrect.	In the source data file, change the data type of the columns to be imported. If this error information is displayed, change the data type to **numeric**.
null value in column "staff_id" violates not-null constraint	The not-null constraint is violated.	In the source data file, add values to the specified columns. If this error information is displayed, add values to the **staff_id** column.
duplicate key value violates unique constraint "reg_id_pk"	The unique constraint is violated.	• Delete the duplicate rows from the source data file. • Run the **SELECT** statement with the **DISTINCT** keyword to ensure that all imported rows are unique. `INSERT INTO reasons SELECT DISTINCT * FROM foreign_tpcds_reasons;`
value too long for type character varying(16)	The column length exceeds the upper limit.	In the source data file, change the column length. If this error information is displayed, reduce the column length to no greater than 16 bytes.

4.2.1.3 Importing ORC or CarbonData Data from OBS

4.2.1.3.1 Preparing Data on OBS

Scenarios

Before you use the SQL on OBS feature to query OBS data:

1. You have stored the ORC data on OBS.

   For example, the ORC table has been created when you use the Hive or Spark component, and the ORC data has been stored on OBS.

   Assume that there are two ORC data files, named **product_info.0** and **product_info.1**, whose original data is stored in the **demo.db/ product_info_orc/** directory of the **mybucket** OBS bucket. You can view their original data in **Original Data**.

2. If your data files are already on OBS, perform steps in **Obtaining the OBS Path of Original Data and Setting Read Permission**.

   📖 NOTE

   This section uses the ORC format as an example to describe how to import data. The method for importing CarbonData data is similar.

Original Data

Assume that you have stored the two ORC data files on OBS and their original data is as follows:

- Data file **product_info.0**

  The file contains the following data:

  ```
  100,XHDK-A-1293-#fJ3,2017-09-01,A,2017 Autumn New Shirt Women,red,M,
  328,2017-09-04,715,good!
  205,KDKE-B-9947-#kL5,2017-09-01,A,2017 Autumn New Knitwear Women,pink,L,
  584,2017-09-05,406,very good!
  300,JODL-X-1937-#pV7,2017-09-01,A,2017 autumn new T-shirt men,red,XL,1245,2017-09-03,502,Bad.
  310,QQPX-R-3956-#aD8,2017-09-02,B,2017 autumn new jacket women,red,L,411,2017-09-05,436,It's
  really super nice.
  150,ABEF-C-1820-#mC6,2017-09-03,B,2017 Autumn New Jeans Women,blue,M,
  1223,2017-09-06,1200,The seller's packaging is exquisite.
  ```

- Data file **product_info.1**

  The file contains the following data:

  ```
  200,BCQP-E-2365-#qE4,2017-09-04,B,2017 autumn new casual pants men,black,L,
  997,2017-09-10,301,The clothes are of good quality.
  250,EABE-D-1476-#oB1,2017-09-10,A,2017 autumn new dress women,black,S,
  841,2017-09-15,299,Follow the store for a long time.
  108,CDXK-F-1527-#pL2,2017-09-11,A,2017 autumn new dress women,red,M,85,2017-09-14,22,It's
  really amazing to buy.
  450,MMCE-H-4728-#nP9,2017-09-11,A,2017 autumn new jacket women,white,M,
  114,2017-09-14,22,Open the package and the clothes have no odor.
  260,OCDA-G-2817-#bD3,2017-09-12,B,2017 autumn new woolen coat women,red,L,
  2004,2017-09-15,826,Very favorite clothes.
  ```

Obtaining the OBS Path of Original Data and Setting Read Permission

Step 1 Log in to the OBS management console.

Click **Service List** and choose **Object Storage Service** to open the OBS management console.

Step 2 Obtain the OBS path for storing source data files.

After the source data files are uploaded to an OBS bucket, a globally unique access path is generated. You need to specify the OBS paths of source data files when creating a foreign table.

For details about how to view the OBS path for storing the backup files, see **Accessing an Object Using Its URL** in *Object Storage Service Console Operation Guide*.

For example, the OBS paths are as follows:

```
https://obs.ap-southeast-1.myhuaweicloud.com/mybucket/demo.db/product_info_orc/product_info.0
https://obs.ap-southeast-1.myhuaweicloud.com/mybucket/demo.db/product_info_orc/product_info.1
```

Step 3 Grant the OBS bucket read permission for the user.

The user who executes the SQL on OBS function needs to obtain the read permission on the OBS bucket where the source data file is located. You can configure the ACL for the OBS buckets to grant the read permission to a specific user.

For details, see **Configuring a Bucket ACL** in *Object Storage Service Console Operation Guide*.

----End

4.2.1.3.2 Creating a Foreign Server

This section describes how to create a foreign server that is used to define the information about OBS servers and is invoked by foreign tables. For details about the syntax for creating foreign servers, see CREATE SERVER.

(Optional) Creating a User and a Database and Granting the User Foreign Table Permissions

Common users do not have permissions to create foreign servers and tables. If you want to use a common user to create foreign servers and tables in a customized database, perform the following steps to create a user and a database, and grant the user foreign table permissions.

In the following example, a common user **dbuser** and a database **mydatabase** are created. Then, an administrator is used to grant foreign table permissions to user **dbuser**.

Step 1 Connect to the default database **gaussdb** as a database administrator through the database client tool provided by GaussDB(DWS).

For example, use the gsql client to connect to the database by running the following command:

```
gsql -d gaussdb -h 192.168.2.30 -U dbadmin -p 8000 -W password -r
```

Step 2 Create a common user and use it to create a database.

Create a user named **dbuser** that has the permission to create databases.

CREATE USER *dbuser* **WITH** *CREATEDB* **PASSWORD** '*password*';

Switch to the created user.
SET ROLE *dbuser* **PASSWORD** '*password*';

Run the following command to create the database demo:
CREATE DATABASE *mydatabase*;

Query the database.

SELECT * FROM pg_database;

The database is successfully created if the returned result contains information about **mydatabase**.

```
datname   | datdba | encoding | datcollate | datctype | datistemplate | datallowconn | datconnlimit |
datlastsysoid | datfrozenxid | dattablespace | datcompatibility |            datacl

-----------+--------+----------+------------+----------+---------------+--------------+--------------+----------------
--------------+---------------+------------------+------------------------------------
--------------
 template1 |    10 |      0 | C        | C       | t          | t          |      -1 |       14146 |      1351
|      1663 | ORA         | {=c/Ruby,Ruby=CTc/Ruby}
 template0 |    10 |      0 | C        | C       | t          | f          |      -1 |       14146 |      1350
|      1663 | ORA         | {=c/Ruby,Ruby=CTc/Ruby}
 gaussdb   |    10 |      0 | C        | C       | f          | t          |      -1 |       14146 |      1352 |
1663 | ORA        | {=Tc/Ruby,Ruby=CTc/Ruby,chaojun=C/Ruby,hu
obinru=C/Ruby}
 mydatabase | 17000 |      0 | C        | C       | f          | t          |      -1 |       14146 |      1351
|      1663 | ORA       |
(4 rows)
```

Step 3 Grant the permissions for creating foreign servers and using foreign tables to a common user as the administrator.

Connect to the new database as a database administrator through the database client tool provided by GaussDB(DWS).

You can use the gsql client to run the following command to switch to an administrator user and connect to the new database:

\c *mydatabase dbadmin*;

Enter the password of the system administrator as prompted.

📖 **NOTE**

> Note that you must use the administrator account to connect to the database where a foreign server is to be created and foreign tables are used; and then grant permissions to the common user.

By default, only system administrators can create foreign servers. Common users can create foreign servers only after being authorized. Run the following command to grant the permission:
GRANT ALL ON SCHEMA public **TO** *dbuser*;
GRANT ALL ON FOREIGN DATA WRAPPER dfs_fdw **TO** *dbuser*;

where *fdw_name* can be **hdfs_fdw** or **dfs_fdw**, and **dbuser** is the name of the user who creates SERVER.

Run the following command to grant the user the permission to use foreign tables:

ALTER USER *dbuser* **USEFT**;

Query for the user.

```
SELECT r.rolname, r.rolsuper, r.rolinherit,
  r.rolcreaterole, r.rolcreatedb, r.rolcanlogin,
  r.rolconnlimit, r.rolvalidbegin, r.rolvaliduntil,
  ARRAY(SELECT b.rolname
      FROM pg_catalog.pg_auth_members m
      JOIN pg_catalog.pg_roles b ON (m.roleid = b.oid)
      WHERE m.member = r.oid) as memberof
, r.rolreplication
, r.rolauditadmin
, r.rolsystemadmin
, r.roluseft
FROM pg_catalog.pg_roles r
ORDER BY 1;
```

The authorization is successful if the **dbuser** information in the returned result contains the UseFT permission.

```
rolname  | rolsuper | rolinherit | rolcreaterole | rolcreatedb | rolcanlogin | rolconnlimit | rolvalidbegin |
rolvaliduntil | memberof | rolreplication | rolauditadmin | rolsystemadmin | roluseft
-----------+----------+------------+---------------+-------------+-------------+-------------+--------------+---------------
+---------------+----------+----------------+---------------+---------------+----------------+----------
 dbuser    | f       | t         | f            | t          | t          |     -1 |            |             | {}    | f
| f          | f          | t
 lily     | f       | t         | f            | f          | t          |     -1 |            |             | {}    | f          |
f          | f          | f
 Ruby     | t       | t         | t            | t          | t          |     -1 |            |             | {}    | t
| t          | t          | t
```

----End

Creating a Foreign Server

Step 1 Use the user who is about to create a foreign server to connect to the corresponding database.

In this example, use common user **dbuser** created in **(Optional) Creating a User and a Database and Granting the User Foreign Table Permissions** to connect to **mydatabase** created by the user. You need to connect to the database through the database client tool provided by GaussDB(DWS).

You can use the **gsql** client to log in to the database in either of the following ways:

- If you have logged in to the gsql client, run the following command to switch the database and user:
  \c *mydatabase dbuser*;

  Enter the password as prompted.

- If you have not logged in to the gsql client or have exited the gsql client by running the **\q** command, run the following command to reconnect to it:
  gsql -d *mydatabase* -h *192.168.2.30* -U *dbuser* -p 8000 -r

  Enter the password as prompted.

Step 2 Create a foreign server.

For details about the syntax for creating foreign servers, see CREATE SERVER.

For example, run the following command to create a foreign server named **obs_server**.

```
CREATE SERVER obs_server FOREIGN DATA WRAPPER dfs_fdw
OPTIONS (
  address 'obs.ap-southeast-1.myhuaweicloud.com' ,
  ACCESS_KEY 'access_key_value_to_be_replaced',
```

```
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced',
encrypt 'on',
type 'obs'
);
```

Mandatory parameters are described as follows:

- *Name of the foreign server*

  You can customize a name.

  In this example, the name is set to **obs_server**.

- **FOREIGN DATA WRAPPER**

  fdw_name can be **hdfs_fdw** or **dfs_fdw**, which already exists in the database.

- **OPTIONS parameters**

  – **address**

    Specifies the endpoint of the OBS service.

    Obtain the address as follows:

    i. Obtain the OBS path by performing **2** in **Preparing Data on OBS**.

    ii. The OBS path on OBS is **obs.***xxx.xxx*.**com**, which is the endpoint of OBS.

  – (Optional) **Access keys (AK and SK)**

    GaussDB(DWS) needs to use the access keys (AK and SK) to access OBS. Therefore, you must obtain the access keys first.

    ▪ (Mandatory) **access_key**: specifies users' AK information.

    ▪ (Mandatory) **secret_access_key**: specifies users' SK information.

    For details about how to obtain the access keys, see **Creating Access Keys (AK and SK)**.

  – **type**

    Its value is **obs**, which indicates that **dfs_fdw** connects to OBS.

Step 3 View the foreign server.

```
SELECT * FROM pg_foreign_server WHERE srvname='obs_server';
```

The server is successfully created if the returned result is as follows:

```
 srvname   | srvowner | srvfdw | srvtype | srvversion | srvacl
|                                                       srvoptions

-----------+----------+--------+---------+------------+--------
+----------------------------------------------------------------------------------------------------------------------
----------------------------------------------
-----------
 obs_server |   24661 | 13686 |         |            |       |
{address=xxx.xxx.x.xxx,access_key=xxxxxxxxxxxxxxxxxxx,type=obs,secret_access_key=xxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxx}
(1 row)
```

----End

4.2.1.3.3 Creating a Foreign Table

After performing steps in **Creating a Foreign Server**, create an OBS foreign table in the GaussDB(DWS) database to access the data stored in OBS. An OBS foreign table is read-only. It can only be queried using **SELECT**.

Creating a Foreign Table

The syntax for creating a foreign table is as follows. For details, see the syntax **CREATE FOREIGN TABLE (SQL on Hadoop or OBS)**.

```
CREATE FOREIGN TABLE [ IF NOT EXISTS ] table_name
( [ { column_name type_name
  [ { [CONSTRAINT constraint_name] NULL |
  [CONSTRAINT constraint_name] NOT NULL |
   column_constraint [...]} ] |
   table_constraint [, ...]} [, ...] ] )
  SERVER dfs_server
  OPTIONS ( { option_name ' value ' } [, ...] )
  DISTRIBUTE BY {ROUNDROBIN | REPLICATION}
  [ PARTITION BY ( column_name ) [ AUTOMAPPED ] ] ;
```

For example, when creating a foreign table named *product_info_ext_obs*, set parameters in the syntax as follows:

- **table_name**

  Specifies the name of the foreign table to be created.

- **Table column definitions**

  - **column_name**: specifies the name of a column in the foreign table.

  - **type_name**: specifies the data type of the column.

  Multiple columns are separate by commas (,).

  The number of fields and field types in the foreign table must be the same as those in the data stored on OBS.

- **SERVER dfs_server**

  This parameter specifies the foreign server name of the foreign table. This server must exist. The foreign server connects to OBS to read data by setting its foreign server.

  Enter the name of the foreign server created by following steps in **Creating a Foreign Server**.

- **OPTIONS parameters**

  These are parameters associated with the foreign table. The key parameters are as follows:

  - **format**: indicates the file format on OBS. The ORC and CARBONDATA formats are supported.

  - **foldername**: This parameter is mandatory. It indicates the OBS path of the data source file. You only need to enter */Bucket name/Folder directory level/*.

    You can perform **2** in **Preparing Data on OBS** to obtain the complete OBS path of the data source file. The path is the endpoint of the OBS service.

  - **totalrows**: This parameter is optional. It does not indicate the total rows of the imported data. Because OBS may store many files, it is slow to analyze data. This parameter allows you to set an estimated value so that the optimizer can estimate the table size according to the value. Generally, query efficiency is relatively high when the estimated value is almost the same as the actual value.

  - **encoding**: encoding of data source files in foreign tables. The default value is **utf8**. This parameter is mandatory for OBS foreign tables.

- **DISTRIBUTE BY**:

  This clause is mandatory. Currently, OBS foreign tables support only the **ROUNDROBIN** distribution mode.

  It indicates that when a foreign table reads data from the data source, each node in the GaussDB(DWS) cluster randomly reads some data and integrates the random data to a complete data set.

- **Other parameters in the syntax**

  Other parameters are optional and can be configured as required. In this example, they do not need to be configured.

Based on the preceding settings, the command for creating the foreign table is as follows:

Create an OBS foreign table that does not contain partition columns. The foreign server associated with the table is **obs_server**, the file format on OBS corresponding to the table is ORC, and the data storage path on OBS is**/mybucket/data/**.

```
DROP FOREIGN TABLE IF EXISTS product_info_ext_obs;
CREATE FOREIGN TABLE product_info_ext_obs
(
    product_price            integer       not null,
    product_id               char(30)      not null,
    product_time             date        ,
    product_level            char(10)       ,
    product_name              varchar(200)   ,
    product_type1             varchar(20)    ,
    product_type2             char(10)       ,
    product_monthly_sales_cnt  integer        ,
    product_comment_time       date          ,
    product_comment_num         integer        ,
    product_comment_content     varchar(200)
) SERVER obs_server
OPTIONS (
format 'orc',
foldername '/mybucket/demo.db/product_info_orc/',
encoding 'utf8',
totalrows '10'
)
DISTRIBUTE BY ROUNDROBIN;
```

Create an OBS foreign table that contains partition columns. The **product_info_ext_obs** foreign table uses the **product_manufacturer** column as the partition key. The following partition directories exist in **obs/mybucket/demo.db/product_info_orc/**:

Partition directory 1: product_manufacturer=10001

Partition directory 2: product_manufacturer=10010

Partition directory 3: product_manufacturer=10086

...
```
DROP FOREIGN TABLE IF EXISTS product_info_ext_obs;
CREATE FOREIGN TABLE product_info_ext_obs
(
    product_price            integer       not null,
    product_id               char(30)      not null,
    product_time             date        ,
    product_level            char(10)       ,
    product_name              varchar(200)   ,
    product_type1             varchar(20)    ,
    product_type2             char(10)       ,
```

```
      product_monthly_sales_cnt    integer     ,
      product_comment_time         date        ,
      product_comment_num           integer       ,
      product_comment_content      varchar(200)   ,
      product_manufacturer     integer
) SERVER obs_server
OPTIONS (
format 'orc',
foldername '/mybucket/demo.db/product_info_orc/',
encoding 'utf8',
totalrows '10'
)
DISTRIBUTE BY ROUNDROBIN
PARTITION BY (product_manufacturer) AUTOMAPPED;
```

4.2.1.3.4 Querying Data on OBS Through Foreign Tables

Viewing Data on OBS by Directly Querying the Foreign Table

If the data amount is small, you can directly run **SELECT** to query the foreign table and view the data on OBS.

Step 1 Run the following command to query data from the foreign table:

```
SELECT * FROM product_info_ext_obs;
```

If the query result is the same as the data in **Original Data**, the import is successful. The following information is displayed at the end of the query result:

```
(10 rows)
```

After data is queried, you can insert the data to common tables in the database.

----End

Querying Data After Importing It

Step 1 Create a table in GaussDB(DWS) to store imported data.

The target table structure must be the same as the structure of the foreign table created in **Creating a Foreign Table**. That is, both tables must have the same number of columns and column types.

For example, create a table named *product_info*. The table example is as follows:

```
DROP TABLE IF EXISTS product_info;

CREATE TABLE product_info
(
    product_price            integer        not null,
    product_id               char(30)       not null,
    product_time              date        ,
    product_level            char(10)       ,
    product_name              varchar(200)   ,
    product_type1             varchar(20)    ,
    product_type2            char(10)       ,
    product_monthly_sales_cnt    integer        ,
    product_comment_time         date         ,
    product_comment_num           integer        ,
    product_comment_content      varchar(200)
)
with (
orientation = column,
compression=middle
)
DISTRIBUTE BY HASH (product_id);
```

Step 2 Run the **INSERT INTO.. SELECT ..** command to import data from the foreign table to the target table.

Example:

```
INSERT INTO product_info SELECT * FROM product_info_ext_obs;
```

If information similar to the following is displayed, the data has been imported.

```
INSERT 0 10
```

Step 3 Run the following **SELECT** command to view data imported from OBS to GaussDB(DWS):

```
SELECT * FROM product_info;
```

If the query result is the same as the data in **Original Data**, the import is successful. The following information is displayed at the end of the query result:

```
(10 rows)
```

----End

4.2.1.3.5 Deleting Resources

After completing operations in this tutorial, if you no longer need to use the resources created during the operations, you can delete them to avoid resource waste or quota occupation. The procedure is as follows:

1. **Deleting the Foreign Table and Target Table**
2. **Deleting the Created Foreign Server**
3. **Deleting the Database and the User to Which the Database Belongs**

   If you have performed steps in **(Optional) Creating a User and a Database and Granting the User Foreign Table Permissions**, delete the database and the user to which the database belongs.

Deleting the Foreign Table and Target Table

Step 1 (Optional) If you have performed steps in **Querying Data After Importing It**, run the following command to delete the target table:

```
DROP TABLE product_info;
```

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

```
DROP TABLE
```

Step 2 Run the following statement to delete the foreign table:

```
DROP FOREIGN TABLE product_info_ext_obs;
```

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

```
DROP FOREIGN TABLE
```

----End

Deleting the Created Foreign Server

Step 1 Use the user who created the foreign server to connect to the database where the foreign server is located.

In this example, common user **dbuser** is used to create the foreign server in **mydatabase**. You need to connect to the database through the database client

tool provided by GaussDB(DWS). You can use the gsql client to log in to the database in either of the following ways:

- If you have logged in to the gsql client, run the following command to switch the database and user:

  \c *mydatabase dbuser*;

  Enter the password as prompted.

- If you have logged in to the gsql client, you can run the **\q** command to exit gsql, and run the following command to reconnect to it:

  gsql -d *mydatabase* -h *192.168.2.30* -U *dbuser* -p 8000 -r

  Enter the password as prompted.

Step 2 Delete the created foreign server.

Run the following command to delete the server. For details about the syntax, see DROP SERVER.

DROP SERVER *obs_server*;

The database is deleted if the following information is displayed:

DROP SERVER

View the foreign server.

SELECT * FROM pg_foreign_server WHERE srvname='*obs_server*';

The server is successfully deleted if the returned result is as follows:

```
 srvname | srvowner | srvfdw | srvtype | srvversion | srvacl | srvoptions
---------+----------+--------+---------+------------+--------+------------
(0 rows)
```

----End

Deleting the Database and the User to Which the Database Belongs

If you have performed steps in **(Optional) Creating a User and a Database and Granting the User Foreign Table Permissions**, perform the following steps to delete the database and the user to which the database belongs.

Step 1 Delete the customized database.

Connect to the default database **gaussdb** through the database client tool provided by GaussDB(DWS).

If you have logged in to the database using the gsql client, run the following command to switch the database and user:

Switch to the default database.

\c gaussdb

Enter your password as prompted.

Run the following command to delete the customized database:

DROP DATABASE *mydatabase*;

The database is deleted if the following information is displayed:

DROP DATABASE

Step 2 Delete the common user created in this example as the administrator.

Connect to the database as a database administrator through the database client tool provided by GaussDB(DWS).

If you have logged in to the database using the **gsql** client, run the following command to switch the database and user:

\c gaussdb dbadmin

Run the following command to reclaim the permission for creating foreign servers:
REVOKE ALL ON FOREIGN DATA WRAPPER dfs_fdw **FROM** *dbuser*;

The name of **FOREIGN DATA WRAPPER** must be **dfs_fdw**. **dbuser** is the username for creating **SERVER**.

Run the following command to delete the user:

DROP USER *dbuser*;

You can run the **\du** command to query for the user and check whether the user has been deleted.

----End

4.2.1.3.6 Supported Data Types

In the big data field, the mainstream file format is ORC, which is supported by GaussDB(DWS). You can use Hive to export data to an ORC file and use a read-only foreign table to query and analyze the data in the ORC file. Therefore, you need to map the data types supported by the ORC file format with the data types supported by GaussDB(DWS). For details, see **Table 1 Mapping between ORC read-only foreign tables and Hive data types**. Similarly, GaussDB(DWS) exports data through a write-only foreign table, and stores the data in the ORC format. Using Hive to read the ORC file content also requires matched data types. **Table 4-6** shows the matching relationship.

Table 4-5 Mapping between ORC read-only foreign tables and Hive data types

Type	Type Supported by GaussDB(DWS) Foreign Tables	Hive Table Type
1-byte integer	TINYINT (not recommended)	TINYINT
	SMALLINT (recommended)	TINYINT
2-byte integer	SMALLINT	SMALLINT
4-byte integer	INTEGER	INT
8-byte integer	BIGINT	BIGINT
Single-precision floating point number	FLOAT4 (REAL)	FLOAT

Type	Type Supported by GaussDB(DWS) Foreign Tables	Hive Table Type
Double-precision floating point number	FLOAT8(DOUBLE PRECISION)	DOUBLE
Scientific data type	DECIMAL[p (,s)] (The maximum precision can reach up to 38.)	DECIMAL (The maximum precision can reach up to 38.) (HIVE 0.11)
Date type	DATE	DATE
Time type	TIMESTAMP	TIMESTAMP
Boolean type	BOOLEAN	BOOLEAN
CHAR type	CHAR(n)	CHAR (n)
VARCHAR type	VARCHAR(n)	VARCHAR (n)
String (large text object)	TEXT(CLOB)	STRING

Table 4-6 Mapping between ORC write-only foreign tables and Hive data types

Type	Type Supported by GaussDB(DWS) Internal Tables (Data Source Table)	Type Supported by GaussDB(DWS) Write-only Foreign Tables	Hive Table Type
1-byte integer	TINYINT	TINYINT (not recommended)	SMALLINT
		SMALLINT (recommended)	SMALLINT
2-byte integer	SMALLINT	SMALLINT	SMALLINT
4-byte integer	INTEGER, BINARY_INTEGER	INTEGER	INT
8-byte integer	BIGINT	BIGINT	BIGINT

Type	Type Supported by GaussDB(DWS) Internal Tables (Data Source Table)	Type Supported by GaussDB(DWS) Write-only Foreign Tables	Hive Table Type
Single-precision floating point number	FLOAT4, REAL	FLOAT4, REAL	FLOAT
Double-precision floating point number	DOUBLE PRECISION, FLOAT8, BINARY_DOUBLE	DOUBLE PRECISION, FLOAT8, BINARY_DOUBLE	DOUBLE
Scientific data type	DECIMAL, NUMERIC	DECIMAL[p (,s)] (The maximum precision can reach up to 38.)	*precision* ≤ **38**: DECIMAL; *precision* > **38**: STRING
Date type	DATE	TIMESTAMP[(p)] [WITHOUT TIME ZONE]	TIMESTAMP
Time type	TIME [(p)] [WITHOUT TIME ZONE], TIME [(p)] [WITH TIME ZONE]	TEXT	STRING
	TIMESTAMP[(p)] [WITHOUT TIME ZONE], TIMESTAMP[(p)][WITH TIME ZONE], SMALLDATETIME	TIMESTAMP[(p)] [WITHOUT TIME ZONE]	TIMESTAMP
	INTERVAL DAY (l) TO SECOND (p), INTERVAL [FIELDS] [(p)]	VARCHAR(n)	VARCHAR(n)
Boolean type	BOOLEAN	BOOLEAN	BOOLEAN
CHAR type	CHAR(n), CHARACTER(n), NCHAR(n)	CHAR(n), CHARACTER(n), NCHAR(n)	n ≤ **255**: CHAR(n); n > **255**: STRING

Type	Type Supported by GaussDB(DWS) Internal Tables (Data Source Table)	Type Supported by GaussDB(DWS) Write-only Foreign Tables	Hive Table Type
VARCHAR type	VARCHAR(n), CHARACTER VARYING(n), VARCHAR2(n)	VARCHAR(n)	$n \le$ **65535**: VARCHAR(n); $n >$ **65535**: STRING
	NVARCHAR2(n)	TEXT	STRING
String (large text object)	TEXT, CLOB	TEXT, CLOB	STRING
Monetary type	MONEY	NUMERIC	BIGINT

NOTICE

1. The GaussDB(DWS) foreign table supports the NULL definition, and the Hive data table supports and uses the corresponding NULL definition.

2. The value range of TINYINT in the Hive data table is [-128, 127], and the value range of TINYINT in GaussDB(DWS) is [0, 255]. You are advised to use the SMALLINT type when creating a GaussDB(DWS) read-only foreign table for TINYINT in the Hive table. If TINYINT is used, the read value may be different from the actual value. Similarly, when exporting data of the TINYINT type from GaussDB(DWS), you are advised to use the SMALLINT type for write-only foreign tables and Hive tables.

3. The time zone definition is not supported by the date and time types of the GaussDB(DWS) foreign table, or by the Hive table.

4. The date type in Hive contains only date. The date type in GaussDB(DWS) contains date and time.

5. In GaussDB(DWS), ORC files can be compressed in ZLIB, SNAPPY, LZ4, or NONE mode.

6. The FLOAT4 format itself is not accurate, and the sum operation results vary with environments. You are advised to use the DECIMAL type in high-precision scenarios.

7. In Teradata-compatible mode, foreign tables do not support the DATE type.

4.2.2 Using GDS to Import Data from a Remote Server

4.2.2.1 Importing Data In Parallel Using GDS

INSERT and **COPY** statements can be used only for serially importing a small volume of data. To import a large volume of data to GaussDB(DWS), you can use GDS to import data in parallel using a foreign table.

In the current GDS version, you can import data to databases from pipe files.

- When the local disk space of the GDS user is insufficient:
  - Data in HDFS can be directly written to a pipe without occupying extra disk space.
- If you need to cleanse data before importing data:
  - You can compile a program as needed and write the data to be processed into a pipe file.

  📖 **NOTE**

  - The current version does not support data import through GDS in SSL mode. Do not use GDS in SSL mode.
  - All pipe files mentioned in this section refer to named pipes on Linux.

Overview

You can import data in parallel from the common file system (excluding HDFS) of a server to GaussDB(DWS).

Data files to be imported are specified based on the import policy and data formats set in a foreign table. Data is imported in parallel through multiple DNs from source data files to the database, which improves the overall data import performance. **Figure 4-4** shows an example.

- The CN only plans data import tasks and delivers the tasks to DNs. In this case, the CN is released to process other tasks.
- In this way, the computing capacities and bandwidths of all the DNs are fully leveraged to import data, improving the import performance.

You can pre-process data (such as invalid character replacement and fault tolerance processing) by setting parameters in a foreign table.

Figure 4-4 Importing data in parallel
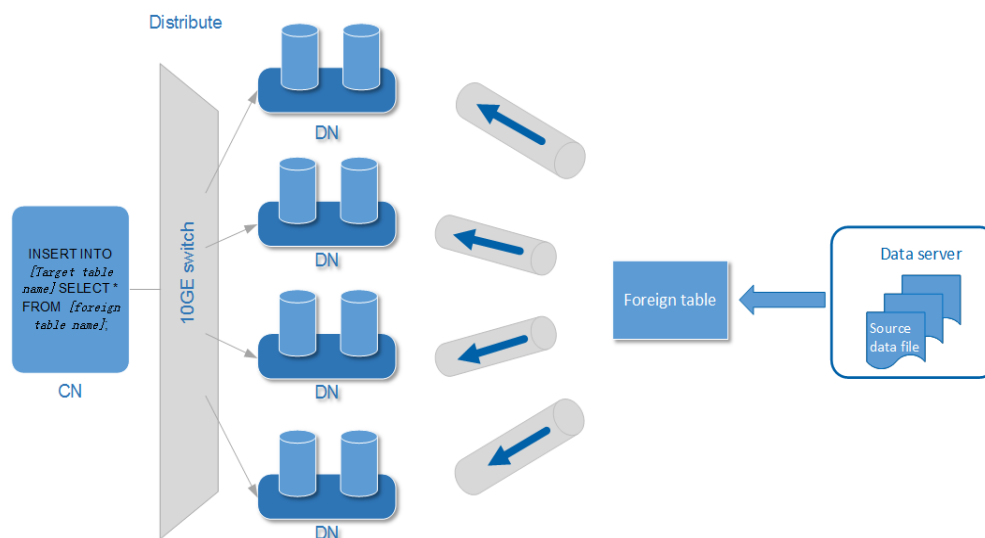
The concepts mentioned in the preceding figure are described as follows:

- **CN**: coordinator of GaussDB(DWS). After receiving import SQL requests from an application or client, the CN plans import tasks and delivers the tasks to DNs.

- **DN (Datanode)**: data node of GaussDB(DWS). After receiving the import tasks delivered by the CN, DNs import data from the source data file to the target table in the database through a foreign table.

- **Source data file**: a file that stores data to be imported.

- **Data server**: a server that stores source data files. For security purposes, it is recommended that the data server and GaussDB(DWS) be on the same intranet.

- **Foreign table**: a table that stores information, such as the source location, format, destination location, encoding format, and data delimiter of a source data file. It is used to associate source data files with the target table.

- **Target table**: a table in the database. It can be a row-store table or column-store table. Data in the source data files will be imported to this table.

Parallel Import Using GDS

- If a large volume of data is stored on multiple servers, deploy, configure, and start GDS on each server. Then, data on all the servers can be imported in parallel, as shown in **Figure 4-5**.

Figure 4-5 Parallel import from multiple data servers
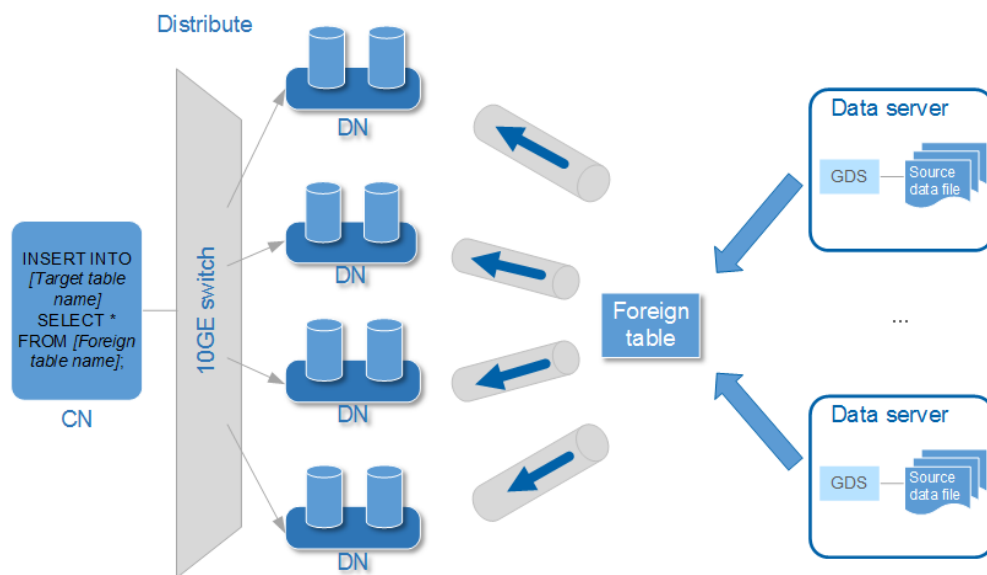
NOTICE

The number of GDS processes cannot exceed that of DNs. If multiple GDS processes are connected to one DN, some of the processes will probably become abnormal.

- If data is stored on data servers, and both GaussDB(DWS) and the data servers have available I/O resources, you can use GDS for multi-thread concurrent import.

  GDS determines the number of threads based on the number of concurrent import transactions. That is, even if multi-thread import is configured before GDS startup, the import of a single transaction will not be accelerated. By default, an **INSERT** statement is an import transaction.

  Multi-thread concurrent import enables you to:

  – Fully use resources and improve the concurrent import efficiency when you import multiple tables to the database.

  – Speed up the import of a table with a large volume of data.

    Table data is split into multiple data files, and multi-thread concurrent import is implemented by importing data using multiple foreign tables at the same time. Ensure that a data file can be read only by one foreign table.

Import Process

Figure 4-6 Concurrent import procedure of GDS
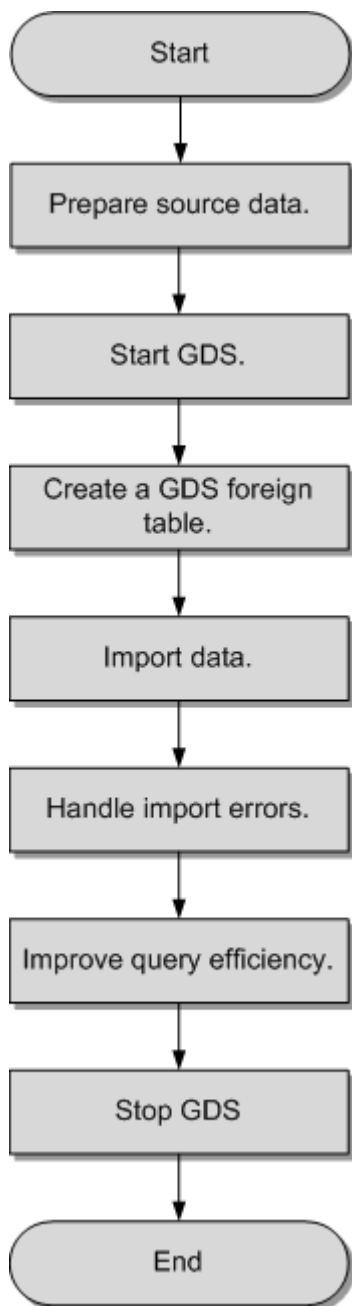
Table 4-7 Process description

Procedure	Description
Prepare source data.	Prepare the source data files to be imported to the database and upload the files to the data server.
	For details, see **Preparing Source Data**.

Procedure	Description
Start GDS.	Install, configure, and enable GDS on the data server. For details, see **Installing, Configuring, and Starting GDS**.
Create a foreign table.	A foreign table is used to identify source files. The foreign table stores information, such as the source location, format, destination location, encoding format, and inter-data delimiter of a source data file. For details, see **Creating a GDS Foreign Table**.
Import data.	After creating the foreign table, run the **INSERT** statement to quickly import data to the target table. For details, see **Importing Data**.
Handle the error table.	If errors occur during parallel data import, handle errors based on the error information to ensure data integrity. For details, see **Handling Import Errors**.
Improve query efficiency.	After data is imported, run the **ANALYZE** statement to generate table statistics. The **ANALYZE** statement stores the statistics in the **PG_STATISTIC** system catalog. The execution plan generator uses the statistics to generate the most efficient query execution plan.
Stop GDS.	After data import is complete, log in to each data server and stop GDS. For details, see **Stopping GDS**.

4.2.2.2 Preparing Source Data

Scenario

Generally, the data to be imported has been uploaded to the data server. In this case, you only need to check the communication between the data server and GaussDB(DWS), and record the data storage directory on the data server before the import.

If the data has not been uploaded to the data server, perform the following operations to upload it:

Procedure

Step 1 Log in to the data server as user **root**.

Step 2 Create the directory **/input_data**.

```
mkdir -p /input_data
```

Step 3 Upload the source data files to the created directory.

GDS parallel import supports source data only in CSV or TEXT format.

----End

4.2.2.3 Installing, Configuring, and Starting GDS

Scenario

GaussDB(DWS) uses GDS to allocate the source data for parallel data import. Deploy GDS on the data server.

If a large volume of data is stored on multiple data servers, install, configure, and start GDS on each server. Then, data on all the servers can be imported in parallel. The procedure for installing, configuring, and starting GDS is the same on each data server. This section describes how to perform this procedure on one data server.

Context

1. GDS can be installed in the following OSs:

   Kunpeng platform:
   - Community Enterprise Operating System 7.6
   - EulerOS 2.0 SP8
   - Red Hat Enterprise Linux Server release 7.5
   - NeoKylin 7.5/7.6

   x86:
   - SUSE Linux Enterprise Server 10 SP4 x86_64
   - SUSE Linux Enterprise Server 11 SP1/SP2/SP3/SP4 x86_64
   - SUSE Linux Enterprise Server 12 SP0/SP1/SP2/SP3/SP5 x86_64
   - Red Hat Enterprise Linux Server release 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
   - Community Enterprise Operating System 6.4/6.5/6.6/6.7/6.8/6.9/7.0/7.1/7.2/7.3/7.4 x86_64
   - EulerOS 2.5 x86_64

2. The GDS version must match the cluster version. For example, GDS V100R008C00 matches DWS 1.3.*X*. Otherwise, the import or export may fail, or the import or export process may fail to respond.

   Therefore, use the latest version of GDS. After the database is upgraded, download the latest version of GaussDB(DWS) GDS as instructed in **Procedure**. When the import or export starts, GaussDB(DWS) checks the GDS versions. If the versions do not match, an error message is displayed and the import or export is terminated.

   To obtain the version number of GDS, run the following command in the GDS decompression directory:

   gds -V

   To view the database version, run the following SQL statement after connecting to the database:

   SELECT version();

Procedure

Step 1 Before using GDS to import or export data, perform the following steps in **Preparing an ECS as the GDS Server and Downloading the GDS Package**.

Step 2 Log in as user **root** to the data server where GDS is to be installed and run the following command to create the directory for storing the GDS package:

mkdir -p */opt/bin/dws*

Step 3 Upload the GDS package to the created directory.

Use the SUSE Linux package as an example. Upload the GDS package **dws_client_8.**x.x**_suse_x64.zip** to the directory created in the previous step.

Step 4 (Optional) If SSL is used, upload the SSL certificates to the directory created in **Step 2**.

Step 5 Go to the directory and decompress the package.

cd */opt/bin/dws*
unzip dws_client_8.x.x_suse_x64.zip

Step 6 Create a GDS user and the user group to which the user belongs. This user is used to start GDS and read source data.

groupadd gdsgrp
useradd -g gdsgrp gds_user

Step 7 Change the owner of the GDS package directory and source data file directory to the GDS user.

chown -R *gds_user:gdsgrp /opt/bin/dws/gds*
chown -R *gds_user:gdsgrp /input_data*

Step 8 Switch to user **gds_user**.

su - *gds_user*

If the current cluster version is 8.0.*x* or earlier, skip **Step 9** and go to **Step 10**.

If the current cluster version is 8.1.*x*, go to the next step.

Step 9 Execute the script on which the environment depends (applicable only to 8.1.*x*).

cd /opt/bin/dws/gds/bin
source gds_env

Step 10 Start GDS.

GDS is green software and can be started after being decompressed. There are two ways to start GDS. One is to run the **gds** command to configure startup parameters. The other is to write the startup parameters into the **gds.conf** configuration file and run the **gds_ctl.py** command to start GDS.

The first method is recommended when you do not need to import data again. The second method is recommended when you need to import data regularly.

- Method 1: Run the **gds** command to start GDS.
  - If data is transmitted in non-SSL mode, run the following command to start GDS:

    gds -d *dir* **-p** *ip:port* **-H** *address_string* **-l** *log_file* **-D -t** *worker_num*

    Example:

    /opt/bin/dws/gds/bin/gds -d */input_data/* **-p** *192.168.0.90:5000* **-H** *10.10.0.1/24* **-l** */opt/bin/dws/gds/gds_log.txt* **-D -t** *2*

  - If data is transmitted in SSL mode, run the following command to start GDS:

    gds -d *dir* **-p** *ip:port* **-H** *address_string* **-l** *log_file* **-D**
    -t *worker_num* **--enable-ssl --ssl-dir** *Cert_file*

    Example:

Run the following command to upload the SSL certificate mentioned in **Step 4** to **/opt/bin**:

```
/opt/bin/dws/gds/bin/gds -d /input_data/ -p 192.168.0.90:5000 -H 10.10.0.1/24 -
l /opt/bin/dws/gds/gds_log.txt -D --enable-ssl --ssl-dir /opt/bin/
```

Replace the information in italic as required.

– **-d** *dir*: directory for storing data files that contain data to be imported. This tutorial uses **/input_data/** as an example.

– **-p** *ip:port*: listening IP address and port for GDS. The default value is **127.0.0.1**. Replace it with the IP address of a 10GE network that can communicate with GaussDB(DWS). The port number ranges from 1024 to 65535. The default port is **8098**. This tutorial uses **192.168.0.90:5000** as an example.

– **-H** *address_string*: specifies the hosts that are allowed to connect to and use GDS. The value must be in CIDR format. Configure this parameter to enable a GaussDB(DWS) cluster to access GDS for data import. Ensure that the network segment covers all hosts in a GaussDB(DWS) cluster.

– **-l** *log_file*: GDS log directory and log file name. This tutorial uses **/opt/bin/dws/gds/gds_log.txt** as an example.

– **-D**: GDS in daemon mode. This parameter is used only in Linux.

– **-t** *worker_num*: number of concurrent GDS threads. If the data server and GaussDB(DWS) have available I/O resources, you can increase the number of concurrent GDS threads.

GDS determines the number of threads based on the number of concurrent import transactions. Even if multi-thread import is configured before GDS startup, the import of a single transaction will not be accelerated. By default, an **INSERT** statement is an import transaction.

– **--enable-ssl**: enables SSL for data transmission.

– **--ssl-dir** *Cert_file*: SSL certificate directory. Set this parameter to the certificate directory in **Step 4**.

– For details about GDS parameters, see **gds**.

● Method 2: Write the startup parameters into the **gds.conf** configuration file and run the **gds_ctl.py** command to start GDS.

a. Run the following command to go to the **config** directory of the GDS package and modify the **gds.conf** configuration file. For details about the parameters in the **gds.conf** configuration file, see **Table 4-8**.

```
vim /opt/bin/dws/gds/config/gds.conf
```

Example:

The **gds.conf** configuration file contains the following information:

```
<?xml version="1.0"?>
<config>
<gds name="gds1" ip="192.168.0.90" port="5000" data_dir="/input_data/" err_dir="/err"
data_seg="100MB" err_seg="100MB" log_file="/log/gds_log.txt" host="10.10.0.1/24"
daemon='true' recursive="true" parallel="32"></gds>
</config>
```

Information in the configuration file is described as follows:

■ The data server IP address is **192.168.0.90** and the GDS listening port is **5000**.

- Data files are stored in the **/input_data/** directory.

- Error log files are stored in the **/err** directory. The directory must be created by a user who has the GDS read and write permissions.

- The size of a single data file is 100 MB.

- The size of a single error log file is 100 MB.

- Logs are stored in the **/log/gds_log.txt** file. The directory must be created by a user who has the GDS read and write permissions.

- Only nodes with the IP address **10.10.0.***can be connected.

- The GDS process is running in daemon mode.

- Recursive data file directories are used.

- The number of concurrent import threads is 2.

b. Start GDS and check whether it has been started.
python3 gds_ctl.py start

Example:

```
cd /opt/bin/dws/gds/bin
python3 gds_ctl.py start
Start GDS gds1            [OK]
gds [options]:
 -d dir          Set data directory.
 -p port          Set GDS listening port.
   ip:port        Set GDS listening ip address and port.
-l log_file       Set log file.
 -H secure_ip_range
                 Set secure IP checklist in CIDR notation. Required for GDS to start.
 -e dir          Set error log directory.
 -E size         Set size of per error log segment.(0 < size < 1TB)
 -S size         Set size of data segment.(1MB < size < 100TB)
 -t worker_num     Set number of worker thread in multi-thread mode, the upper limit is 32. If
without setting, the default value is 1.
 -s status_file    Enable GDS status report.
 -D              Run the GDS as a daemon process.
 -r              Read the working directory recursively.
 -h              Display usage.
```

----End

gds.conf Parameter Description

Table 4-8 gds.conf configuration description

Attribute	Description	Value Range
name	Identifier	-
ip	Listening IP address	The IP address must be valid. Default value: **127.0.0.1**
port	Listening port	Value range: 1024 to 65535 (integer) Default value: **8098**

Attribute	Description	Value Range
data_dir	Data file directory	-
err_dir	Error log file directory	Default value: data file directory
log_file	Log file Path	-
host	Host IP address allowed to be connected to GDS (The value must in CIDR format and this parameter is available for the Linux OS only.)	-
recursive	Whether the data file directories are recursive	Value range: • **true**: recursive • **false**: not recursive Default value: **false**
daemon	Whether the process is running in daemon mode	Value range: • **true**: The process is running in daemon mode. • **false**: The process is not running in daemon mode. Default value: **false**
parallel	Number of concurrent data import threads	Value range: 0 to 32 (integer) Default value: **1**

4.2.2.4 Creating a GDS Foreign Table

The source data information and GDS access information are configured in a foreign table. Then, GaussDB(DWS) can import data from a data server to a database table based on the configuration in the foreign table.

Procedure

Step 1 Collect source data information and GDS access information.

You need to collect the following source data information:

- **format**: format of the data to be imported. Only data in CSV, TEXT, or FIXED format can be imported using GDS in parallel.
- **header**: whether a source data file has a header. This parameter is set only for files in CSV or FIXED format.
- **delimiter**: delimiter in the source data file. For example, it can be a comma (,).
- **encoding**: encoding format of the data source file. Assume that the encoding format is UTF-8.

- **eol**: line break character in the data file. It can be a default character, such as 0x0D0A or 0X0A, or a customized line break character, such as a string: **!@#**. This parameter can be set only for TEXT import.
- For details about more source data information configured in a foreign table, see data format parameters.

You need to collect the following GDS access information:

location: GDS URL. GDS information in **Installing, Configuring, and Starting GDS** is used as an example. In non-SSL mode, **location** is set to **gsfs**:// *192.168.0.90:5000|/input_data/*. In SSL mode, **location** is set to **gsfss**:// *192.168.0.90:5000|/input_data/*. **192.168.0.90:5000** indicates the IP address and port number of GDS. **input_data** indicates the path of data source files managed by GDS. Replace the values as required.

Step 2 Design an error tolerance mechanism for data import.

GaussDB(DWS) supports the following error tolerance in data import:

- **fill_missing_fields**: This parameter specifies whether to report an error when the last column in a row of the source data file is empty, or to fill the column with **NULL**.
- **ignore_extra_data**: This parameter specifies whether to report an error when the number of columns in the source data file is greater than that specified in the foreign table, or to ignore the extra columns.
- **per node reject_limit**: This parameter specifies the number of data format errors allowed on each DN. If the number of errors recorded in the error table on a DN exceeds the specified value, the import will fail and an error message will be reported. You can also set it to **unlimited**.
- **compatible_illegal_chars**: This parameter specifies whether to report an error when an illegal character is encountered, or to convert it and proceed with the import.

  The following describes the rules for converting an illegal character:
  - **\0** is converted to a space.
  - Other illegal characters are converted to question marks.
  - If **NULL**, **DELIMITER**, **QUOTE**, or **ESCAPE** is also set to a space or question mark, an error message such as "illegal chars conversion may confuse COPY escape 0x20" is displayed, prompting you to modify parameter settings that may cause import errors.

- **error_table_name**: This parameter specifies the name of the table that records data format errors. After the parallel import, you can query this table for error details.
- **remote log 'name'**: This parameter specifies whether to store data format errors in files on the GDS server. **name** is the prefix of the error data file.
- For details about more error tolerance parameters, see error tolerance parameters.

Step 3 After connecting to the database using **gsql** or Data Studio, create a GDS foreign table based on the collected and design information.

For example:

```
CREATE FOREIGN TABLE foreign_tpcds_reasons
(
```

```
   r_reason_sk integer not null,
   r_reason_id char(16) not null,
   r_reason_desc char(100)
)
 SERVER gsmpp_server
 OPTIONS
(
LOCATION 'gsfs://192.168.0.90:5000/* | gsfs://192.168.0.91:5000/*',
FORMAT 'CSV' ,
DELIMITER ',',
ENCODING 'utf8',
HEADER 'false',
FILL_MISSING_FIELDS 'true',
IGNORE_EXTRA_DATA 'true'
)
LOG INTO product_info_err
PER NODE REJECT LIMIT 'unlimited';
```

The following describes information in the preceding command:

- The columns specified in the foreign table must be the same as those in the target table.

- Retain the value **gsmpp_server** for **SERVER**.

- Set **location** based on the GDS access information collected in **Step 1**. If SSL is used, replace **gsfs** with **gsfss**.

- Set **FORMAT**, **DELIMITER**, **ENCODING**, and **HEADER** based on the source data information collected in **Step 1**.

- Set **FILL_MISSING_FIELDS**, **IGNORE_EXTRA_DATA**, **LOG INTO**, and **PER NODE REJECT LIMIT** based on the error tolerance mechanism designed in **Step 2**. **LOG INTO** specifies the name of the error table.

For details about the CREATE FOREIGN TABLE syntax, see CREATE FOREIGN TABLE (for GDS Import and Export).

----End

Example

For more examples, see **Example of Importing Data Using GDS**.

- Example 1: Create a GDS foreign table named **foreign_tpcds_reasons**. The data format is CSV.
```
CREATE FOREIGN TABLE foreign_tpcds_reasons
(
  r_reason_sk integer not null,
  r_reason_id char(16) not null,
  r_reason_desc char(100)
)
 SERVER gsmpp_server OPTIONS (location 'gsfs://192.168.0.90:5000/* | gsfs://192.168.0.91:5000/*',
FORMAT 'CSV',MODE 'Normal', ENCODING 'utf8', DELIMITER E'\x08', QUOTE E'\x1b', NULL '');
```

- Example 2: Create a GDS foreign table named **foreign_tpcds_reasons_SSL**. SSL is used and the data format is CSV.
```
CREATE FOREIGN TABLE foreign_tpcds_reasons_SSL
(
  r_reason_sk integer not null,
  r_reason_id char(16) not null,
  r_reason_desc char(100)
)
 SERVER gsmpp_server OPTIONS (location 'gsfss://192.168.0.90:5000/* | gsfss://192.168.0.91:5000/*',
FORMAT 'CSV',MODE 'Normal', ENCODING 'utf8', DELIMITER E'\x08', QUOTE E'\x1b', NULL '');
```

- Example 3: Create a GDS foreign table named **foreign_tpcds_reasons**. The data format is TEXT.

```
CREATE FOREIGN TABLE foreign_tpcds_reasons
(
  r_reason_sk integer not null,
  r_reason_id char(16) not null,
  r_reason_desc char(100)
) SERVER gsmpp_server OPTIONS (location 'gsfs://192.168.0.90:5000/* | gsfs://192.168.0.91:5000/*',
FORMAT 'TEXT', delimiter E'\x08',  null '',reject_limit '2',EOL '0x0D') WITH err_foreign_tpcds_reasons;
```

4.2.2.5 Importing Data

This section describes how to create tables in GaussDB(DWS) and import data to the tables.

Before importing all the data from a table containing over 10 million records, you are advised to import some of the data, and check whether there is data skew and whether the distribution keys need to be changed (for details, see **Checking for Data Skew**). Troubleshoot the data skew if any. It is costly to address data skew and change the distribution keys after a large amount of data has been imported.

Prerequisites

The GDS server can communicate with GaussDB(DWS).

- You need to create an ECS as the GDS server.
- The created ECS and GaussDB(DWS) must belong to the same region, VPC, and subnet.

Procedure

Step 1 Create a table in GaussDB(DWS) to store imported data. For details, see CREATE TABLE.

Step 2 (Optional) If the target table has an index, the index information is incrementally updated during the import, affecting data import performance. You are advised to delete the indexes of related tables before importing data. If the data uniqueness cannot be ensured, you are not advised to delete the unique indexes. You can create indexes again after the import is complete.

1. Assume that the ordinary index **product_idx** exists in the **product_id** column of the target table **product_info**. Delete the index from the table before importing data.
   ```
   DROP INDEX product_idx;
   ```

2. After importing the data, create the **reasons_idx** index again.
   ```
   CREATE INDEX product_idx ON product_info(product_id);
   ```

Step 3 Import data.

INSERT INTO *[Target table name]* **SELECT * FROM** *[Foreign table name]*

- If information similar to the following is displayed, the data has been imported. Query the error information table to check whether any data format errors occurred. For details, see **Handling Import Errors**.
  ```
  INSERT 0 9
  ```

- If data fails to be loaded, troubleshoot the problem by following the instructions provided in **Handling Import Errors** and try again.

📖 **NOTE**

- If a data loading error occurs, the entire data import task will fail.
- Compile a batch-processing task script to concurrently import data. The degree of parallelism (DOP) depends on the server resource usage. You can test-import several tables, monitor resource utilization, and increase or reduce concurrency accordingly. Common resource monitoring commands include **top** for monitoring memory and CPU usage, **iostat** for monitoring I/O usage, and **sar** for monitoring networks. For details about application cases, see .
- If possible, more GDS servers can significantly improve the data import efficiency. For details about application cases, see **Parallel Import from Multiple Data Servers**.
- In a scenario where many GDS servers import data concurrently, you can increase the TCP Keepalive interval for connections between GDS servers and DNs to ensure connection stability. (The recommended interval is 5 minutes.) TCP Keepalive settings of the cluster affect its fault detection response time.

----End

Example:

1. Create a target table named **reasons**.
   ```
   CREATE TABLE reasons
   (
     r_reason_sk   integer  not null,
     r_reason_id   char(16) not null,
     r_reason_desc char(100)
   )
   DISTRIBUTE BY HASH (r_reason_sk);
   ```

2. You are advised to delete the indexes from the target table before the import.

   Assume that an ordinary table index **reasons_idx** exists on the **r_reason_id** column in the **reasons** table. Delete the index before the import. Delete the index from the table.
   ```
   DROP INDEX reasons_idx;
   ```

3. Import data from source data files through the **foreign_tpcds_reasons** foreign table to the **reasons** table.
   ```
   INSERT INTO reasons SELECT * FROM foreign_tpcds_reasons ;
   ```

4. You can create indexes again after the import is complete.
   ```
   CREATE INDEX reasons_idx ON reasons(r_reasons_id);
   ```

4.2.2.6 Handling Import Errors

Scenarios

Handle errors that occurred during data import.

Querying Error Information

Errors that occur when data is imported are divided into data format errors and non-data format errors.

- Data format error

  When creating a foreign table, specify **LOG INTO** *error_table_name*. Data format errors occurring during the data import will be written into the specified table. You can run the following SQL statement to query error details:

```
SELECT * FROM error_table_name;
```

Table 4-9 lists the columns of the *error_table_name* table.

Table 4-9 Columns in the *error_table_name* table

Column	Type	Description
nodeid	integer	ID of the node where an error is reported
begintime	timestamp with time zone	Time when a data format error is reported
filename	character varying	Name of the source data file where a data format error occurs If you use GDS for importing data, the error information includes the IP address and port number of the GDS server.
rownum	bigint	Number of the row where an error occurs in a source data file
rawrecord	text	Raw record of the data format error in the source data file
detail	text	Error details

- Non-data format error

  A non-data format error leads to the failure of an entire data import task. You can locate and troubleshoot a non-data format error based on the error message displayed during data import.

Handling data import errors

Troubleshoot data import errors based on obtained error information and the description in the following table.

Table 4-10 Handling data import errors

Error Information	Cause	Solution
missing data for column "r_reason_desc"	1. The number of columns in the source data file is less than that in the foreign table. 2. In a TEXT format source data file, an escape character (for example, \) leads to delimiter or quote mislocation. Example: The target table contains three columns as shown in the following command output. The escape character (\) converts the delimiter (|) into the value of the second column, causing loss of the value of the third column. BE|Belgium\\\|1	1. If an error is reported due to missing columns, perform the following operations: ● Add the **r_reason_desc** column to the source data file. ● When creating a foreign table, set the parameter **fill_missing_fields** to **on**. In this way, if the last column of a row in the source data file is missing, it is set to **NULL** and no error will be reported. 2. Check whether the row where an error occurred contains the escape character (\). If the row contains such a character, you are advised to set the parameter **noescaping** to **true** when creating a foreign table, indicating that the escape character (\) and the characters following it are not escaped.
extra data after last expected column	The number of columns in the source data file is greater than that in the foreign table.	● Delete the unnecessary columns from the source data file. ● When creating a foreign table, set the parameter **ignore_extra_data** to **on**. In this way, if the number of columns in a source data file is greater than that in the foreign table, the extra columns at the end of rows will not be imported.

Error Information	Cause	Solution
invalid input syntax for type numeric: "a"	The data type is incorrect.	In the source data file, change the data type of the columns to be imported. If this error information is displayed, change the data type to **numeric**.
null value in column "staff_id" violates not-null constraint	The not-null constraint is violated.	In the source data file, add values to the specified columns. If this error information is displayed, add values to the **staff_id** column.
duplicate key value violates unique constraint "reg_id_pk"	The unique constraint is violated.	- Delete the duplicate rows from the source data file. - Run the **SELECT** statement with the **DISTINCT** keyword to ensure that all imported rows are unique. **INSERT INTO** reasons **SELECT DISTINCT * FROM** foreign_tpcds_reasons;
value too long for type character varying(16)	The column length exceeds the upper limit.	In the source data file, change the column length. If this error information is displayed, reduce the column length to no greater than 16 bytes (VARCHAR2).

4.2.2.7 Stopping GDS

Scenarios

Stop GDS after data is imported successfully.

Procedure

Step 1 Log in as user **gds_user** to the data server where GDS is installed.

Step 2 Select the mode of stopping GDS based on the mode of starting it.

- If GDS is started using the **gds** command, perform the following operations to stop GDS:

  a. Query the GDS process ID:
     ```
     ps -ef|grep gds
     ```
     For example, the GDS process ID is 128954.
     ```
     ps -ef|grep gds
     gds_user 128954    1 0 15:03 ?        00:00:00 gds -d /input_data/ -p 192.168.0.90:5000 -l /log/
     ```

```
gds_log.txt -D
gds_user 129003 118723  0 15:04 pts/0    00:00:00 grep gds
```

    b.   Run the **kill** command to stop GDS. **128954** in the command is the GDS process ID.

```
kill -9 128954
```

----End

4.2.2.8 Example of Importing Data Using GDS

Parallel Import from Multiple Data Servers

The data servers and the cluster reside on the same intranet. The IP addresses are **192.168.0.90** and **192.168.0.91**. Source data files are in CSV format.

1. Create the target table **tpcds.reasons**.

```
CREATE TABLE tpcds.reasons
(
  r_reason_sk integer not null,
  r_reason_id char(16) not null,
  r_reason_desc char(100)
);
```

2. Log in to each GDS data server as user **root** and create the **/input_data** directory for storing data files on the servers. The following takes the data server whose IP address is **192.168.0.90** as an example. Operations on the other server are the same.

```
mkdir -p /input_data
```

3. (Optional) Create a user and the user group it belongs to. The user is used to start GDS. If the user and user group exist, skip this step.

```
groupadd gdsgrp
useradd -g gdsgrp gds_user
```

4. Evenly distribute source data files to the **/input_data** directories on the data servers.

5. Change the owners of source data files and the **/input_data** directory on each data server to **gds_user**. The data server whose IP address is **192.168.0.90** is used as an example.

```
chown -R gds_user:gdsgrp /input_data
```

6. Log in to each data server as user **gds_user** and start GDS.

    The GDS installation path is **/opt/bin/dws/gds**. Source data files are stored in **/input_data/**. The IP addresses of the data servers are **192.168.0.90** and **192.168.0.91**. The GDS listening port is **5000**. GDS runs in daemon mode.

    Start GDS on the data server whose IP address is **192.168.0.90**.

```
/opt/bin/dws/gds/bin/gds -d /input_data -p 192.168.0.90:5000 -H 10.10.0.1/24 -D
```

    Start GDS on the data server whose IP address is **192.168.0.91**.

```
/opt/bin/dws/gds/bin/gds -d /input_data -p 192.168.0.91:5000 -H 10.10.0.1/24  -D
```

7. Create the foreign table **tpcds.foreign_tpcds_reasons** for receiving data from the data server.

    Data export mode settings are as follows:

    – Set the import mode to **Normal**.

    – When GDS is started, the source data file directory is **/input_data** and the GDS listening port is **5000**. Therefore, set **location** to **gsfs://192.168.0.90:5000/* | gsfs://192.168.0.91:5000/***.

Information about the data format is configured based on data format parameters specified during data export. The parameter configurations are as follows:

- **format** is set to **CSV**.
- **encoding** is set to **UTF-8**.
- **delimiter** is set to **E'\x08'**.
- **quote** is set to **E'\x1b'**.
- **null** is set to an empty string without quotation marks.
- **escape** defaults to the value of **quote**.
- **header** is set to **false**, indicating that the first row is identified as a data row in an imported file.

Configure import error tolerance parameters as follows:

- Set **PER NODE REJECT LIMIT** (number of allowed data format errors) to **unlimited**. In this case, all the data format errors detected during data import will be tolerated.
- Set **LOG INTO** to **err_tpcds_reasons**. The data format errors detected during data import will be recorded in the **err_tpcds_reasons** table.

Based on the above settings, the foreign table is created using the following statement:

```
CREATE FOREIGN TABLE tpcds.foreign_tpcds_reasons
(
  r_reason_sk integer not null,
  r_reason_id char(16) not null,
  r_reason_desc char(100)
)
SERVER gsmpp_server OPTIONS (location 'gsfs://192.168.0.90:5000/* | gsfs://192.168.0.91:5000/*',
format 'CSV',mode 'Normal', encoding 'utf8', delimiter E'\x08', quote E'\x1b', null '', fill_missing_fields
'false') LOG INTO err_tpcds_reasons PER NODE REJECT LIMIT 'unlimited';
```

8. Import data through the foreign table **tpcds.foreign_tpcds_reasons** to the target table **tpcds.reasons**.
```
INSERT INTO tpcds.reasons SELECT * FROM tpcds.foreign_tpcds_reasons;
```

9. Query data import errors in the **err_tpcds_reasons** table and rectify the errors (if any). For details, see **Handling Import Errors**.
```
SELECT * FROM err_tpcds_reasons;
```

10. After data import is complete, log in to each data server as user **gds_user** and stop GDS.

The data server whose IP address is **192.168.0.90** is used as an example. The GDS process ID is **128954**.
```
ps -ef|grep gds
gds_user 128954     1  0 15:03 ?        00:00:00 gds -d /input_data -p 192.168.0.90:5000 -D
gds_user 129003 118723  0 15:04 pts/0    00:00:00 grep gds
kill -9 128954
```

Data Import Using Multiple Threads

The data servers and the cluster reside on the same intranet. The server IP address is **192.168.0.90**. Source data files are in CSV format. Data will be imported to two tables using multiple threads in **Normal** mode.

1. In the database, create the target tables **tpcds.reasons1** and **tpcds.reasons2**.
```
CREATE TABLE tpcds.reasons1
(
  r_reason_sk integer not null,
```

```
    r_reason_id char(16) not null,
    r_reason_desc char(100)
) ;
CREATE TABLE tpcds.reasons2
(
    r_reason_sk integer not null,
    r_reason_id char(16) not null,
    r_reason_desc char(100)
) ;
```

2. Log in to the GDS data server as user **root**, and then create the data file directory **/input_data** and its sub-directories **/input_data/import1/** and **/input_data/import2/**.

   mkdir -p */input_data*

3. Store the source data files of the target table **tpcds.reasons1** in **/input_data/import1/** and the source data files of the target table **tpcds.reasons2** in **/input_data/import2/**.

4. (Optional) Create a user and the user group it belongs to. The user is used to start GDS. If the user and user group already exist, skip this step.

   groupadd gdsgrp
   useradd -g gdsgrp gds_user

5. Change the owners of source data files and the **/input_data** directory on the data server to **gds_user**.

   chown -R *gds_user:gdsgrp /input_data*

6. Log in to the data server as user **gds_user** and start GDS.

   The GDS installation path is **/gds**. Source data files are stored in **/input_data/**. The IP address of the data server is **192.168.0.90**. The GDS listening port is **5000**. GDS runs in daemon mode. The degree of parallelism is 2. A recursive directory is specified.

   /gds/gds -d */input_data* **-p** *192.168.0.90:5000* **-H** 10.10.0.1/24 **-D -t** 2 **-r**

7. In the database, create the foreign tables **tpcds.foreign_tpcds_reasons1** and **tpcds.foreign_tpcds_reasons2** for the source data.

   The foreign table **tpcds.foreign_tpcds_reasons1** is used as an example to describe how to configure parameters in a foreign table.

   Data export mode settings are as follows:

   – Set the import mode to **Normal**.

   – When GDS is started, the configured source data file directory is **/input_data** and the GDS listening port is **5000**. However, source data files are actually stored in **/input_data/import1/**. Therefore, set **location** to **gsfs://192.168.0.90:5000/import1/***.

   Information about the data format is configured based on data format parameters specified during data export. The parameter configurations are as follows:

   – **format** is set to **CSV**.

   – **encoding** is set to **UTF-8**.

   – **delimiter** is set to **E'\x08'**.

   – **quote** is set to **E'\x1b'**.

   – **null** is set to an empty string without quotation marks.

   – **escape** defaults to the value of **quote**.

   – **header** is set to **false**, indicating that the first row is identified as a data row in an imported file.

Configure import error tolerance parameters as follows:

- Set **PER NODE REJECT LIMIT** (number of allowed data format errors) to **unlimited**. In this case, all the data format errors detected during data import will be tolerated.

- Set **LOG INTO** to **err_tpcds_reasons1**. The data format errors detected during data import will be recorded in the **err_tpcds_reasons1** table.

- If the last column of a source data file is missing, the **fill_missing_fields** parameter is automatically set to **NULL**.

Based on the preceding settings, the foreign table **tpcds.foreign_tpcds_reasons1** is created using the following statement:

```
CREATE FOREIGN TABLE tpcds.foreign_tpcds_reasons1
(
  r_reason_sk integer not null,
  r_reason_id char(16) not null,
  r_reason_desc char(100)
) SERVER gsmpp_server OPTIONS (location 'gsfs://192.168.0.90:5000/import1/*', format 'CSV',mode
'Normal', encoding 'utf8', delimiter E'\x08', quote E'\x1b', null '',fill_missing_fields 'on')LOG INTO
err_tpcds_reasons1 PER NODE REJECT LIMIT 'unlimited';
```

Based on the preceding settings, the foreign table **tpcds.foreign_tpcds_reasons2** is created using the following statement:

```
CREATE FOREIGN TABLE tpcds.foreign_tpcds_reasons2
(
  r_reason_sk integer not null,
  r_reason_id char(16) not null,
  r_reason_desc char(100)
) SERVER gsmpp_server OPTIONS (location 'gsfs://192.168.0.90:5000/import2/*', format 'CSV',mode
'Normal', encoding 'utf8', delimiter E'\x08', quote E'\x1b', null '',fill_missing_fields 'on')LOG INTO
err_tpcds_reasons2 PER NODE REJECT LIMIT 'unlimited';
```

8. Import data through the foreign table **tpcds.foreign_tpcds_reasons1** to **tpcds.reasons1** and through **tpcds.foreign_tpcds_reasons2** to **tpcds.reasons2**.
```
INSERT INTO tpcds.reasons1 SELECT * FROM tpcds.foreign_tpcds_reasons1;
INSERT INTO tpcds.reasons2 SELECT * FROM tpcds.foreign_tpcds_reasons2;
```

9. Query data import errors in the **err_tpcds_reasons1** and **err_tpcds_reasons2** tables and rectify the errors (if any). For details, see **Handling Import Errors**.
```
SELECT * FROM err_tpcds_reasons1;
SELECT * FROM err_tpcds_reasons2;
```

10. After data import is complete, log in to the data server as user **gds_user** and stop GDS.

The GDS process ID is **128954**.
```
ps -ef|grep gds
gds_user 128954     1  0 15:03 ?        00:00:00 gds -d /input_data -p 192.168.0.90:5000 -D -t 2 -r
gds_user 129003 118723  0 15:04 pts/0    00:00:00 grep gds
kill -9 128954
```

Importing Data Through a Pipe File

Step 1 Start the GDS.
```
gds -d /***/gds_data/ -D -p 192.168.0.1:7789 -l /***/gds_log/aa.log -H 0/0 -t 10 -D
```

If you need to set the timeout interval of a pipe, use the **--pipe-timeout** parameter.

Step 2 Import data.

1. Log in to the database and create an internal table.
```
CREATE TABLE test_pipe_1( id integer not null, sex text not null, name  text );
```

    2. Create a read-only foreign table.
```
CREATE FOREIGN TABLE foreign_test_pipe_tr( like test_pipe ) SERVER gsmpp_server OPTIONS
(LOCATION 'gsfs://192.168.0.1:7789/foreign_test_pipe.pipe', FORMAT 'text', DELIMITER ',',  NULL '',
EOL '0x0a' ,file_type 'pipe',auto_create_pipe 'false');
```

    3. Execute the import statement. The statement is blocked.
```
INSERT INTO test_pipe_1 select * from foreign_test_pipe_tr;
```

Step 3 Import data through the GDS pipes.

    1. Log in to the GDS server and go to the GDS data directory.
```
cd /***/gds_data/
```

    2. Create a pipe. If **auto_create_pipe** is set to **true**, skip this step.
```
mkfifo foreign_test_pipe.pipe;
```

       📖 **NOTE**

          A pipe will be automatically cleared after an operation is complete. To perform
          another operation, create a pipe file again.

    3. Write data to the pipe.
```
cat postgres_public_foreign_test_pipe_tw.txt > foreign_test_pipe.pipe
```

    4. To read the compressed file to the pipe, run the following command.
```
gzip -d < out.gz > foreign_test_pipe.pipe
```

    5. To read the HDFS file to the pipe, run the following command.
```
hdfs dfs -cat - /user/hive/***/test_pipe.txt > foreign_test_pipe.pipe
```

Step 4 View the result returned by the import statement.
```
INSERT INTO test_pipe_1 select * from foreign_test_pipe_tr;
INSERT 0 4
SELECT * FROM test_pipe_1;
id | sex |      name
----+-----+----------------
3 | 2   | 11111111111111
1 | 2   | 11111111111111
2 | 2   | 11111111111111
4 | 2   | 11111111111111
(4 rows)
```

----End

Importing Data Through Multi-Process Pipes

GDS also supports importing data through multi-process pipes. That is, one foreign table corresponds to multiple GDSs.

The following takes importing a local file as an example.

Step 1 Start multiple GDSs. If the GDSs have been started, skip this step.
```
gds -d /***/gds_data/ -D -p 192.168.0.1:7789 -l /***/gds_log/aa.log -H 0/0 -t 10 -D
gds -d /***/gds_data_1/ -D -p 192.168.0.1:7790 -l /***/gds_log_1/aa.log -H 0/0 -t 10 -D
```

If you need to set the timeout interval of a pipe, use the **--pipe-timeout** parameter.

Step 2 Import data.

    1. Log in to the database and create an internal table.
```
CREATE TABLE test_pipe( id integer not null, sex text not null, name  text );
```

    2. Create a read-only foreign table.
```
CREATE FOREIGN TABLE foreign_test_pipe_tr( like test_pipe ) SERVER gsmpp_server OPTIONS
(LOCATION 'gsfs://192.168.0.1:7789/foreign_test_pipe.pipe|gsfs://192.168.0.1:7790/
foreign_test_pipe.pipe', FORMAT 'text', DELIMITER ',', NULL '', EOL '0x0a' , file_type 'pipe',
auto_create_pipe 'false');
```

3. Execute the import statement. The statement is blocked.
```
INSERT INTO test_pipe_1 select * from foreign_test_pipe_tr;
```

Step 3 Import data through the GDS pipes.

1. Log in to GDS and go to each GDS data directory.
```
cd /***/gds_data/
cd /***/gds_data_1/
```

2. Create a pipe. If **auto_create_pipe** is set to **true**, skip this step.
```
mkfifo foreign_test_pipe.pipe;
```

3. Read each pipe and write the new file to the pipes.
```
cat postgres_public_foreign_test_pipe_tw.txt > foreign_test_pipe.pipe
```

Step 4 View the result returned by the import statement.
```
INSERT INTO test_pipe_1 select * from foreign_test_pipe_tr;
INSERT 0 4
SELECT * FROM test_pipe_1;
id | sex |      name
----+-----+----------------
3 | 2   | 11111111111111
1 | 2   | 11111111111111
2 | 2   | 11111111111111
4 | 2   | 11111111111111
(4 rows)
```

----End

Direct Data Import Between Clusters

Step 1 Start the GDS. (If the process has been started, skip this step.)
```
gds -d /***/gds_data/ -D -p GDS_IP:GDS_PORT -l /***/gds_log/aa.log -H 0/0 -t 10 -D
```

If you need to set the timeout interval of a pipe, use the **--pipe-timeout** parameter.

Step 2 Export data from the source database.

1. Log in to the target database, create an internal table, and write data to the table.
```
CREATE TABLE test_pipe( id integer not null, sex text not null, name  text );
INSERT INTO test_pipe values(1,2,'11111111111111');
INSERT INTO test_pipe values(2,2,'11111111111111');
INSERT INTO test_pipe values(3,2,'11111111111111');
INSERT INTO test_pipe values(4,2,'11111111111111');
INSERT INTO test_pipe values(5,2,'11111111111111');
```

2. Create a write-only foreign table.
```
CREATE FOREIGN TABLE foreign_test_pipe( id integer not null, age text not null, name  text ) SERVER
gsmpp_server OPTIONS (LOCATION 'gsfs://GDS_IP:GDS_PORT/', FORMAT 'text', DELIMITER ',', NULL '',
EOL '0x0a' ,file_type 'pipe') WRITE ONLY;
```

3. Execute the import statement. The statement is blocked.
```
INSERT INTO foreign_test_pipe SELECT * FROM test_pipe;
```

Step 3 Import data to the target cluster.

1. Create an internal table.
```
CREATE TABLE test_pipe (id integer not null, sex text not null, name text);
```

2. Create a read-only foreign table.
```
CREATE FOREIGN TABLE foreign_test_pipe(like test_pipe) SERVER gsmpp_server OPTIONS (LOCATION
'gsfs://GDS_IP:GDS_PORT/', FORMAT 'text', DELIMITER ',', NULL '', EOL '0x0a' , file_type 'pipe',
auto_create_pipe 'false');
```

3. Run the following command to import data to the table.
```
INSERT INTO test_pipe SELECT * FROM foreign_test_pipe;
```

Step 4 View the result returned by the import statement from the target cluster.

```
SELECT * FROM test_pipe;
 id | sex |      name
----+-----+----------------
  3 | 2   | 11111111111111
  6 | 2   | 11111111111111
  7 | 2   | 11111111111111
  1 | 2   | 11111111111111
  2 | 2   | 11111111111111
  4 | 2   | 11111111111111
  5 | 2   | 11111111111111
  8 | 2   | 11111111111111
  9 | 2   | 11111111111111
(9 rows)
```

----End

NOTE

By default, the pipeline file exported from or imported to GDS is named in the format of *Database name_Schema name_Foreign table name* **.pipe**. Therefore, the database name and schema name of the target cluster must be the same as those of the source cluster. If the database or schema is inconsistent, you can specify the same pipe file in the URL of the **location**.

Example:

- Pipe name specified by a write-only foreign table.
  ```
  CREATE FOREIGN TABLE foreign_test_pipe(id integer not null, age text not null, name  text)
  SERVER gsmpp_server OPTIONS (LOCATION 'gsfs://GDS_IP:GDS_PORT/foreign_test_pipe.pipe',
  FORMAT 'text', DELIMITER ',',  NULL '', EOL '0x0a' ,file_type 'pipe') WRITE ONLY;
  ```

- Pipe name specified by a read-only foreign table.
  ```
  CREATE FOREIGN TABLE foreign_test_pipe(like test_pipe) SERVER gsmpp_server OPTIONS
  (LOCATION 'gsfs://GDS_IP:GDS_PORT/foreign_test_pipe.pipe', FORMAT 'text', DELIMITER ',',  NULL
  '', EOL '0x0a' ,file_type 'pipe',auto_create_pipe 'false');
  ```

4.2.3 Importing Data from MRS to a Cluster

4.2.3.1 Overview

MRS is a big data cluster running based on the open-source Hadoop ecosystem. It provides the industry's latest cutting-edge storage and analytical capabilities of massive volumes of data, satisfying your data storage and processing requirements. For details, see the *MapReduce Service User Guide*.

You can use Hive/Spark (analysis cluster of MRS) to store massive volumes of service data. Hive/Spark data files are stored on HDFS. On GaussDB(DWS), you can connect a GaussDB(DWS) cluster to an MRS cluster, read data from HDFS files, and write the data to GaussDB(DWS) when the clusters are on the same network.

> **NOTICE**
>
> Ensure that MRS can communicate with DWS:
>
> Scenario 1: If MRS and DWS are in the same region and VPC, they can communicate with each other by default.
>
> Scenario 2: If MRS and GaussDB(DWS) are in the same region but in different VPCs, you need to create a VPC peering connection. For details, see **VPC Peering Connection Overview**.
>
> Scenario 3: If MRS and GaussDB(DWS) are not in the same region, you need to use **Cloud Connect (CC)** to create network connections. For details, see the user guide of the corresponding service.
>
> Scenario 4: If MRS is deployed on-premises, you need to use **Direct Connect (DC)** or **Virtual Private Network (VPN)** to connect the network. For details, see the User Guide of the corresponding service.

Importing Data from MRS to a GaussDB(DWS) Cluster

1. **Preparing Data in an MRS Cluster**
2. (Optional) **Manually Creating a Foreign Server**
3. **Creating a Foreign Table**
4. **Importing Data**
5. **Deleting Resources**

4.2.3.2 Preparing Data in an MRS Cluster

Before importing data from MRS to a GaussDB(DWS) cluster, you must have:

1. Created an MRS cluster.
2. Created a Hive/Spark ORC table in the MRS cluster and stored the table data to the HDFS path corresponding to the table.

If you have completed the preparations, skip this section.

In this tutorial, the Hive ORC table will be created in the MRS cluster as an example to complete the preparation work. The process and the SQL syntax for creating a Spark ORC table in the MRS cluster are similar to those in Hive.

Data File

The sample data of the **product_info.txt** data file is as follows:

```
100,XHDK-A-1293-#fJ3,2017-09-01,A,2017 Autumn New Shirt Women,red,M,328,2017-09-04,715,good
205,KDKE-B-9947-#kL5,2017-09-01,A,2017 Autumn New Knitwear Women,pink,L,584,2017-09-05,406,very
good!
300,JODL-X-1937-#pV7,2017-09-01,A,2017 autumn new T-shirt men,red,XL,1245,2017-09-03,502,Bad.
310,QQPX-R-3956-#aD8,2017-09-02,B,2017 autumn new jacket women,red,L,411,2017-09-05,436,It's really
super nice
150,ABEF-C-1820-#mC6,2017-09-03,B,2017 Autumn New Jeans Women,blue,M,1223,2017-09-06,1200,The
seller's packaging is exquisite
200,BCQP-E-2365-#qE4,2017-09-04,B,2017 autumn new casual pants men,black,L,997,2017-09-10,301,The
clothes are of good quality.
250,EABE-D-1476-#oB1,2017-09-10,A,2017 autumn new dress women,black,S,841,2017-09-15,299,Follow
the store for a long time.
108,CDXK-F-1527-#pL2,2017-09-11,A,2017 autumn new dress women,red,M,85,2017-09-14,22,It's really
```

amazing to buy
450,MMCE-H-4728-#nP9,2017-09-11,A,2017 autumn new jacket women,white,M,114,2017-09-14,22,Open the package and the clothes have no odor
260,OCDA-G-2817-#bD3,2017-09-12,B,2017 autumn new woolen coat women,red,L, 2004,2017-09-15,826,Very favorite clothes
980,ZKDS-J-5490-#cW4,2017-09-13,B,2017 Autumn New Women's Cotton Clothing,red,M, 112,2017-09-16,219,The clothes are small
98,FKQB-I-2564-#dA5,2017-09-15,B,2017 autumn new shoes men,green,M,4345,2017-09-18,5473,The clothes are thick and it's better this winter.
150,DMQY-K-6579-#eS6,2017-09-21,A,2017 autumn new underwear men,yellow, 37,2840,2017-09-25,5831,This price is very cost effective
200,GKLW-l-2897-#wQ7,2017-09-22,A,2017 Autumn New Jeans Men,blue,39,5879,2017-09-25,7200,The clothes are very comfortable to wear
300,HWEC-L-2531-#xP8,2017-09-23,A,2017 autumn new shoes women,brown,M,403,2017-09-26,607,good
100,IQPD-M-3214-#yQ1,2017-09-24,B,2017 Autumn New Wide Leg Pants Women,black,M, 3045,2017-09-27,5021,very good.
350,LPEC-N-4572-#zX2,2017-09-25,B,2017 Autumn New Underwear Women,red,M,239,2017-09-28,407,The seller's service is very good
110,NQAB-O-3768-#sM3,2017-09-26,B,2017 autumn new underwear women,red,S, 6089,2017-09-29,7021,The color is very good
210,HWNB-P-7879-#tN4,2017-09-27,B,2017 autumn new underwear women,red,L,3201,2017-09-30,4059,I like it very much and the quality is good.
230,JKHU-Q-8865-#uO5,2017-09-29,C,2017 Autumn New Clothes with Chiffon Shirt,black,M, 2056,2017-10-02,3842,very good

Creating a Hive ORC Table in an MRS Cluster

1. Create an MRS cluster.

    For details, see section **Buying a Custom Cluster** in *MapReduce Service Management Guide*.

2. Download the client.

    a. Go back to the MRS cluster page. Click the cluster name. On the **Dashboard** tab page of the cluster details page, click **Access Manager**. If a message is displayed indicating that EIP needs to be bound, bind an EIP first.

    b. Enter the username **admin** and its password for logging in to MRS Manager. The password is the one you entered when creating the MRS cluster.

    c. Choose **Services** > **Download Client**. Set **Client Type** to **Only configuration files** and set **Download To** to **Server**. Click **OK**.
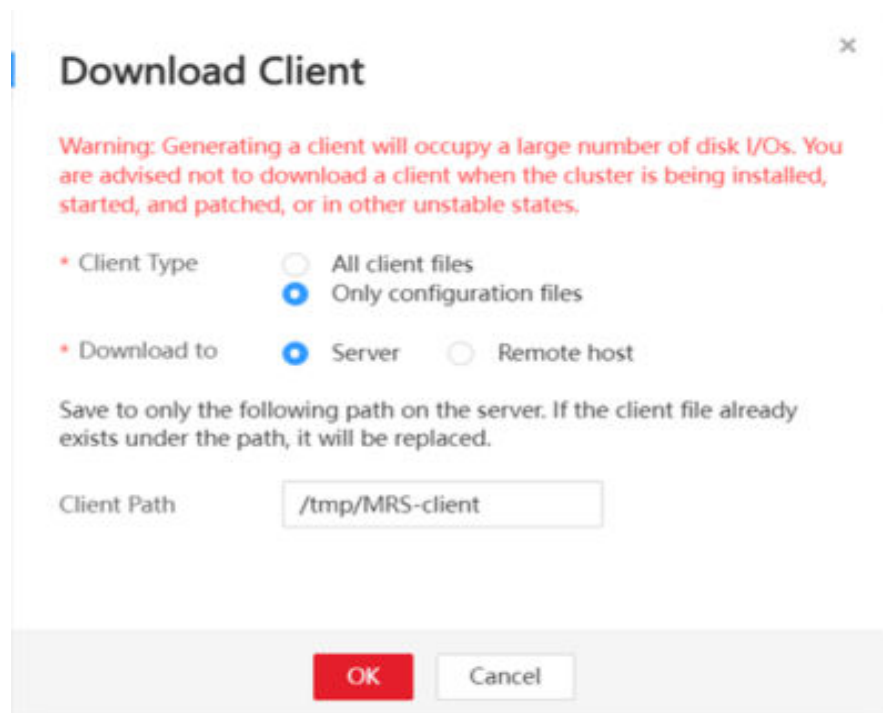
3. Log in to the Hive client of the MRS cluster.

   a. Log in to a Master node.

      For details, see **Logging In to an ECS** in the *MapReduce Service User Guide*.

   b. Run the following command to switch the user:
      ```
      sudo su - omm
      ```

   c. Run the following command to go to the client directory:
      ```
      cd /opt/client
      ```

   d. Run the following command to configure the environment variables:
      ```
      source bigdata_env
      ```

   e. If Kerberos authentication is enabled for the current cluster, run the following command to authenticate the current user. The current user must have the permission to create Hive tables.

      For details, see **Creating a Role** in the *MapReduce Service User Guide*.

   f. Configure roles with the corresponding permissions.

      For details, see **Creating a User** in the *MapReduce Service User Guide*.

   g. Bind roles to users. If the Kerberos authentication is disabled for the current cluster, skip this step.
      ```
      kinit MRS cluster user
      ```
      Example: **kinit hiveuser**

   h. Run the following command to start the Hive client:
      ```
      beeline
      ```

4. Create a database demo on Hive.

   Run the following command to create the database demo:

```
CREATE DATABASE demo;
```

5. Create table **product_info** of the **Hive TEXTFILE** type in the database demo and import the **Data File** (**product_info.txt**) to the HDFS path corresponding to the table.

   Run the following command to switch to the database demo:

   ```
   USE demo;
   ```

   Run the following command to create table **product_info** and define the table fields based on data in the **Data File**.

   ```
   DROP TABLE product_info;

   CREATE TABLE product_info
   (
       product_price            int          ,
       product_id               char(30)     ,
       product_time             date         ,
       product_level            char(10)     ,
       product_name             varchar(200) ,
       product_type1            varchar(20)  ,
       product_type2            char(10)     ,
       product_monthly_sales_cnt int         ,
       product_comment_time     date         ,
       product_comment_num      int          ,
       product_comment_content  varchar(200)
   )
   row format delimited fields terminated by ','
   stored as TEXTFILE;
   ```

   For details about how to import data to an MRS cluster, see **Managing Data Files** in the *MapReduce Service User Guide*.

6. Create a Hive ORC table named **product_info_orc** in the database demo.

   Run the following command to create the Hive ORC table **product_info_orc**. The table fields are the same as those of the **product_info** table created in the previous step.

   ```
   DROP TABLE product_info_orc;

   CREATE TABLE product_info_orc
   (
       product_price            int          ,
       product_id               char(30)     ,
       product_time             date         ,
       product_level            char(10)     ,
       product_name             varchar(200) ,
       product_type1            varchar(20)  ,
       product_type2            char(10)     ,
       product_monthly_sales_cnt int         ,
       product_comment_time     date         ,
       product_comment_num      int          ,
       product_comment_content  varchar(200)
   )
   row format delimited fields terminated by ','
   stored as orc;
   ```

7. Insert data in the **product_info** table to the Hive ORC table **product_info_orc**.

   ```
   insert into product_info_orc select * from product_info;
   ```

   Query table **product_info_orc**.

   ```
   select * from product_info_orc;
   ```

   If data displayed in the **Data File** can be queried, the data has been successfully inserted to the ORC table.

4.2.3.3 Manually Creating a Foreign Server

In the syntax **CREATE FOREIGN TABLE (SQL on Hadoop or OBS)** for creating a foreign table, you need to specify a foreign server associated with the MRS data source connection.

When you create an MRS data source connection on the GaussDB(DWS) management console, the database administrator dbadmin automatically creates a foreign server in the default database **postgres**. If you want to create a foreign table in the default database **postgres** to read MRS data, skip this section.

To allow a common user to create a foreign table in a user-defined database to read MRS data, you must manually create a foreign server in the user-defined database. This section describes how does a common user create a foreign server in a user-defined database. The procedure is as follows:

1. Ensure that an MRS data source connection has been created for the GaussDB(DWS) cluster.

   For details, see section **Creating an MRS Data Source Connection** in the *Data Warehouse Service Management Guide*.

2. **Creating a User and a Database and Granting the User Foreign Table Permissions**

3. **Manually Creating a Foreign Server**

> 📖 **NOTE**
>
> If you no longer need to read data from the MRS data source and have deleted the MRS data source on the GaussDB(DWS) management console, only the foreign server automatically created in the default database **postgres** will be deleted, and the manually created foreign server needs to be deleted manually. For details about the deletion, see **Deleting the Manually Created Foreign Server**.

Creating a User and a Database and Granting the User Foreign Table Permissions

In the following example, a common user **dbuser** and a database **mydatabase** are created. Then, an administrator is used to grant foreign table permissions to user **dbuser**.

Step 1 Connect to the default database **postgres** as a database administrator through the database client tool provided by GaussDB(DWS).

For example, use the **gsql** client to connect to the database by running the following command:

```
gsql -d gaussdb -h 192.168.2.30 -U dbadmin -p 8000 -W password -r
```

Step 2 Create a common user and use it to create a database.

Create a user named **dbuser** that has the permission to create databases.

```
CREATE USER dbuser WITH CREATEDB PASSWORD 'password';
```

Switch to the created user.
```
SET ROLE dbuser PASSWORD 'password';
```

Run the following command to create a database:
```
CREATE DATABASE mydatabase;
```

Query the database.

SELECT * FROM pg_database;

The database is successfully created if the returned result contains information about **mydatabase**.

```
datname   | datdba | encoding | datcollate | datctype | datistemplate | datallowconn | datconnlimit |
datlastsysoid | datfrozenxid | dattablespace | datcompatibility |                datacl

------------+--------+----------+------------+----------+---------------+--------------+--------------+--------------
+--------------+--------------+------------------+---------------------------------------
--------------
 template1 |   10 |      0 | C         | C        | t            | t          |         -1 |        14146 |      1351
|       1663 | ORA         | {=c/Ruby,Ruby=CTc/Ruby}
 template0 |   10 |      0 | C         | C        | t            | f          |         -1 |        14146 |      1350
|       1663 | ORA         | {=c/Ruby,Ruby=CTc/Ruby}
 gaussdb   |   10 |      0 | C         | C        | f            | t          |         -1 |        14146 |      1352 |
1663 | ORA         | {=Tc/Ruby,Ruby=CTc/Ruby,chaojun=C/Ruby,hu
obinru=C/Ruby}
 mydatabase | 17000 |      0 | C         | C        | f            | t          |         -1 |        14146 |      1351
|       1663 | ORA         |
(4 rows)
```

Step 3 Grant the permissions for creating foreign servers and using foreign tables to a common user as the administrator.

Use the connection to create a database as a database administrator.

You can use the **gsql** client to run the following command, switching to an administrator user, and connect to the new database:

\c *mydatabase dbadmin*;

Enter the password as prompted.

📖 **NOTE**

> Note that you must use the administrator account to connect to the database where a foreign server is to be created and foreign tables are used; and then grant permissions to the common user.

By default, only system administrators can create foreign servers. Common users can create foreign servers only after being authorized. Run the following command to grant the permission:

GRANT ALL ON FOREIGN DATA WRAPPER hdfs_fdw **TO** *dbuser*;

The name of **FOREIGN DATA WRAPPER** must be **hdfs_fdw**. **dbuser** is the username for creating **SERVER**.

Run the following command to grant the user the permission to use foreign tables:

ALTER USER *dbuser* **USEFT**;

Query for the user.

```
SELECT r.rolname, r.rolsuper, r.rolinherit,
  r.rolcreaterole, r.rolcreatedb, r.rolcanlogin,
  r.rolconnlimit, r.rolvalidbegin, r.rolvaliduntil,
  ARRAY(SELECT b.rolname
      FROM pg_catalog.pg_auth_members m
      JOIN pg_catalog.pg_roles b ON (m.roleid = b.oid)
      WHERE m.member = r.oid) as memberof
, r.rolreplication
, r.rolauditadmin
, r.rolsystemadmin
```

```
, r.roluseft
FROM pg_catalog.pg_roles r
ORDER BY 1;
```

The authorization is successful if the **dbuser** information in the returned result contains the **UseFT** permission.

```
rolname  | rolsuper | rolinherit | rolcreaterole | rolcreatedb | rolcanlogin | rolconnlimit | rolvalidbegin |
rolvaliduntil | memberof | rolreplication | rolauditadmin | rolsystemadmin | roluseft
-----------+----------+------------+---------------+-------------+-------------+--------------+---------------
+---------------+----------+----------------+---------------+----------------+----------
 dbuser   | f      | t       | f         | t       | t       |     -1 |         |          |{}    | f
| f       | f        | t
 lily     | f      | t       | f         | f       | t       |     -1 |         |          |{}    | f       |
f       | f        | f
 Ruby     | t      | t       | t         | t       | t       |     -1 |         |          |{}    | t
| t        | t           | t
```

----End

Manually Creating a Foreign Server

Step 1 Connect to the default database **postgres** as a database administrator through the database client tool provided by GaussDB(DWS).

You can use the **gsql** client to log in to the database in either of the following ways:

You can use either of the following methods to create the connection:

- If you have logged in to the gsql client, run the following command to switch the database and user:
  ```
  \c postgres dbadmin;
  ```
  Enter the password as prompted.

- If you have not logged in to the gsql client or have exited the gsql client by running the **\q** command, run the following command to reconnect to it:
  ```
  gsql -d postgres -h 192.168.2.30 -U dbadmin -p 8000 -W password -r
  ```

Step 2 Run the following command to query the information about the foreign server that is automatically created:
```
SELECT * FROM pg_foreign_server;
```

The returned result is as follows:

```
            srvname                    | srvowner | srvfdw | srvtype | srvversion | srvacl |            srvoptions
-----------------------------------------------+----------+--------+---------+------------+--------
+------------------------------------------------------------------------------------------------------------------
 gsmpp_server                                  |    10 | 13673 |      |       |    |
 gsmpp_errorinfo_server                        |    10 | 13678 |      |       |    |
 hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca |   16476 | 13685 |      |       |    |
{"address=192.168.1.245:25000,192.168.1.218:25000",hdfscfgpath=/MRS/8f79ada0-
d998-4026-9020-80d6de2692ca,type=hdfs}
(3 rows)
```

In the query result, each row contains the information about a foreign server. The foreign server associated with the MRS data source connection contains the following information:

- The value of **srvname** contains **hdfs_server** and the ID of the MRS cluster, which is the same as the MRS ID in the cluster list on the MRS management console.

- The **address** parameter in the **srvoptions** field contains the IP addresses and ports of the active and standby nodes in the MRS cluster.

You can find the foreign server you want based on the above information and record the values of its **srvname** and **srvoptions**.

Step 3 Switch to the user who is about to create a foreign server to connect to the corresponding database.

In this example, run the following command to use common user **dbuser** created in **Creating a User and a Database and Granting the User Foreign Table Permissions** to connect to **mydatabase** created by the user:

\c *mydatabase dbuser*;

Step 4 Create a foreign server.

For details about the syntax for creating foreign servers, see CREATE SERVER. For example:

CREATE SERVER *hdfs_server_8f79ada0_d998_4026_9020_80d6de2692cahdfs_server* **FOREIGN DATA WRAPPER** HDFS_FDW
OPTIONS
(
address *'192.168.1.245:25000,192.168.1.218:25000'*,
hdfscfgpath *'/MRS/8f79ada0-d998-4026-9020-80d6de2692ca'*,
type 'hdfs'
);

Mandatory parameters are described as follows:

- *Name of the foreign server*

  You can customize a name.

  In this example, specify the name to the value of the **srvname** field recorded in **Step 2**, such as *hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca*.

  Resources in different databases are isolated. Therefore, the names of foreign servers in different databases can be the same.

- **FOREIGN DATA WRAPPER**

  This parameter can only be set to **HDFS_FDW**, which already exists in the database.

- **OPTIONS** parameters

  Set the following parameters to the values under **srvoptions** recorded in **Step 2**.

  – address

    Specifies the IP address and port number of the primary and standby nodes of the HDFS cluster.

  – hdfscfgpath

    Specifies the configuration file path of the HDFS cluster. This parameter is available only when **type** is **HDFS**. You can set only one path.

  – type

    Its value is **hdfs**, which indicates that **HDFS_FDW** connects to HDFS.

Step 5 View the foreign server.

SELECT * FROM pg_foreign_server WHERE
srvname='*hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca*';

The server is successfully created if the returned result is as follows:

```
            srvname              | srvowner | srvfdw | srvtype | srvversion | srvacl
|                                srvoptions
-----------------------------------------------+----------+--------+---------+------------+--------
+--------------------------------------------------------------------------------------------------------
 hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca |   16476 | 13685 |        |           |      |
{"address=192.168.1.245:25000,192.168.1.218:25000",hdfscfgpath=/MRS/8f79ada0-
d998-4026-9020-80d6de2692ca,type=hdfs}
(1 row)
```

----End

4.2.3.4 Creating a Foreign Table

This section describes how to create a Hadoop foreign table in the GaussDB(DWS) database to access the Hadoop structured data stored on MRS HDFS. A Hadoop foreign table is read-only. It can only be queried using **SELECT**.

Prerequisites

- You have created an MRS cluster and imported data to the ORC table in the Hive/Spark database.

  For details, see **Preparing Data in an MRS Cluster**.

- You have created an MRS data source connection for the GaussDB(DWS) cluster.

  For details, see section **Creating an MRS Data Source Connection** in the *Data Warehouse Service Management Guide*.

Obtaining the HDFS Path of the MRS Data Source

There are two methods for you to obtain the HDFS path.

- **Method 1**

  For Hive data, log in to the Hive client of MRS (see **2**), run the following command to view the detailed information about the table, and record the data storage path in the **location** parameter:

  ```
  use <database_name>;
  desc formatted <table_name>;
  ```

  For example, if the value of the **location** parameter in the returned result is **hdfs://hacluster/user/hive/warehouse/demo.db/product_info_orc/**, the HDFS path is **/user/hive/warehouse/demo.db/product_info_orc/**.

- **Method 2**

  Perform the following steps to obtain the HDFS path:

  a. Log in to the MRS management console.

  b. Choose **Cluster** > **Active Cluster** and click the name of the cluster to be queried to enter the page displaying the cluster's basic information.

  c. Click **File Management** and select **HDFS File List**.

  d. Go to the storage directory of the data to be imported to the GaussDB(DWS) cluster and record the path.

Figure 4-7 Checking the data storage path on MRS
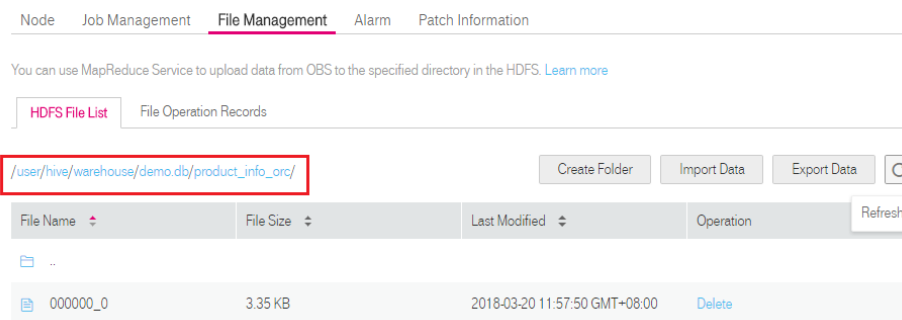
Obtaining Information About the Foreign Server Connected to the MRS Data Source

Step 1 Use the user who creates the foreign server to connect to the corresponding database.

Determine whether to use a common user to create a foreign table in the customized database based on requirements.

- **Yes**

    a. Ensure that you have created the common user **dbuser** and its database **mydatabase**, and manually created a foreign server in **mydatabase** by following steps in **Manually Creating a Foreign Server**.

    b. Connect to the database **mydatabase** as user **dbuser** through the database client tool provided by GaussDB(DWS).

    If you have connected to the database using the gsql client, run the following command to switch the user and database:

    \c *mydatabase dbuser*;

    Enter your password as prompted.

- **No**

    When you create an MRS data source connection on the GaussDB(DWS) management console, the database administrator **dbadmin** automatically creates a foreign server in the default database **postgres**. If you create a foreign table in the default database **postgres** as the database administrator **dbadmin**, you need to connect to the database using the database client tool provided by GaussDB(DWS). For example, use the **gsql** client to connect to the database by running the following command:

    gsql -d postgres -h *192.168.2.30* -U dbadmin -p 8000 -W *password* -r

Step 2 Run the following command to view the information about the created foreign server connected to the MRS data source:

SELECT * FROM pg_foreign_server;

> 📖 **NOTE**
>
> You can also run the **\desc+** command to view the information about the foreign server.

The returned result is as follows:

```
                srvname                | srvowner | srvfdw | srvtype | srvversion | srvacl | srvoptions
---------------------------------------+----------+--------+---------+------------+--------
```

```
+------------------------------------------------------------------------------------------------------------------------
gsmpp_server                               |   10 | 13673 |         |           |       |
gsmpp_errorinfo_server                     |   10 | 13678 |         |           |       |
hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca | 16476 | 13685 |         |           |       |
{"address=192.168.1.245:25000,192.168.1.218:25000",hdfscfgpath=/MRS/8f79ada0-
d998-4026-9020-80d6de2692ca,type=hdfs}
(3 rows)
```

In the query result, each row contains the information about a foreign server. The foreign server associated with the MRS data source connection contains the following information:

- The value of **srvname** contains **hdfs_server** and the ID of the MRS cluster, which is the same as the MRS ID in the cluster list on the MRS management console.

- The **address** parameter in the **srvoptions** field contains the IP addresses and ports of the active and standby nodes in the MRS cluster.

You can find the foreign server you want based on the above information and record the values of its **srvname** and **srvoptions**.

----End

Creating a Foreign Table

After **Obtaining Information About the Foreign Server Connected to the MRS Data Source** and **Obtaining the HDFS Path of the MRS Data Source** are completed, you can create a foreign table to read data from the MRS data source.

The syntax for creating a foreign table is as follows. For details, see the syntax **CREATE FOREIGN TABLE (SQL on Hadoop or OBS)**.

```
CREATE FOREIGN TABLE [ IF NOT EXISTS ] table_name
( [ { column_name type_name
  [ { [CONSTRAINT constraint_name] NULL |
  [CONSTRAINT constraint_name] NOT NULL |
   column_constraint [...]} ] |
   table_constraint [, ...]} [, ...] ] )
 SERVER dfs_server
 OPTIONS ( { option_name ' value ' } [, ...] )
 DISTRIBUTE BY {ROUNDROBIN | REPLICATION}
 [ PARTITION BY ( column_name ) [ AUTOMAPPED ] ] ;
```

For example, when creating a foreign table named *foreign_product_info*, set parameters in the syntax as follows:

- **table_name**

  Mandatory. This parameter specifies the name of the foreign table to be created.

- Table column definitions

  - **column_name**: specifies the name of a column in the foreign table.

  - **type_name**: specifies the data type of the column.

  Multiple columns are separate by commas (,).

  The number of columns and column types in the foreign table must be the same as those in the data stored on MRS. Learn **Data Type Conversion** before defining column data types.

- **SERVER dfs_server**

This parameter specifies the foreign server name of the foreign table. This server must exist. The foreign table can read data from an MRS cluster by configuring the foreign server and connecting to the MRS data source.

Enter the value of the **srvname** field queried in **Obtaining Information About the Foreign Server Connected to the MRS Data Source**.

- **OPTIONS** parameters

  These are parameters associated with the foreign table. The key parameters are as follows:

  - **format**: This parameter is mandatory. The value can only be **orc**. It specifies the format of the source data file. Only Hive ORC files are supported.

  - **foldername**: This parameter is mandatory. It specifies the HDFS directory for storing data or data file path.

    If the MRS analysis cluster has enabled Kerberos authentication, ensure that the MRS user having the MRS data source connection has the read and write permissions for the directory.

    Follow the steps in **Obtaining the HDFS Path of the MRS Data Source** to obtain the HDFS path, which is the value of parameter **foldername**.

  - **encoding**: This parameter is optional. It specifies the encoding format of a source data file in the foreign table. Its default value is **utf8**.

  - **DISTRIBUTE BY**

    This parameter specifies the data read mode for the foreign table. There are two read modes supported. In this example, **ROUNDROBIN** is selected.

    - **ROUNDROBIN**: When a foreign table reads data from the data source, each node in a GaussDB(DWS) cluster randomly reads some data and integrates the random data to a complete data set.

    - **REPLICATION**: When a foreign table reads data from the data source, each node in the GaussDB(DWS) cluster reads a complete data set.

  - Other parameters in the syntax

    Other parameters are optional and can be configured as required. In this example, they do not need to be configured.

Based on the above settings, the foreign table is created using the following statements:

```
DROP FOREIGN TABLE IF EXISTS foreign_product_info;

CREATE FOREIGN TABLE foreign_product_info
(
    product_price              integer        ,
    product_id                 char(30)       ,
    product_time               date           ,
    product_level              char(10)       ,
    product_name               varchar(200)   ,
    product_type1              varchar(20)    ,
    product_type2              char(10)       ,
    product_monthly_sales_cnt  integer        ,
    product_comment_time       date           ,
    product_comment_num        integer        ,
    product_comment_content    varchar(200)
```

```
) SERVER hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca
OPTIONS (
format 'orc',
encoding 'utf8',
foldername '/user/hive/warehouse/demo.db/product_info_orc/'
)
DISTRIBUTE BY ROUNDROBIN;
```

Data Type Conversion

Data is imported to Hive/Spark and then stored on HDFS in ORC format. Actually, GaussDB(DWS) reads ORC files on HDFS, and queries and analyzes data in these files.

Data types supported by Hive/Spark are different from those supported by GaussDB(DWS). Therefore, you need to learn the mapping between them. **Table 4-11** describes the mapping in detail.

Table 4-11 Data type mapping

Type	Column Type Supported by an HDFS/OBS Foreign Table of GaussDB(DWS)	Column Type Supported by a Hive Table	Column Type Supported by a Spark Table
Integer in two bytes	SMALLINT	SMALLINT	SMALLINT
Integer in four bytes	INTEGER	INT	INT
Integer in eight bytes	BIGINT	BIGINT	BIGINT
Single-precision floating point number	FLOAT4 (REAL)	FLOAT	FLOAT
Double-precision floating point number	FLOAT8(DOUBLE PRECISION)	DOUBLE	FLOAT
Scientific data type	DECIMAL[p (,s)] The maximum precision can reach up to 38.	DECIMAL The maximum precision can reach up to 38 (Hive 0.11).	DECIMAL
Date type	DATE	DATE	DATE
Time type	TIMESTAMP	TIMESTAMP	TIMESTAMP
BOOLEAN type	BOOLEAN	BOOLEAN	BOOLEAN

Type	Column Type Supported by an HDFS/OBS Foreign Table of GaussDB(DWS)	Column Type Supported by a Hive Table	Column Type Supported by a Spark Table
CHAR type	CHAR(n)	CHAR (n)	STRING
VARCHAR type	VARCHAR(n)	VARCHAR (n)	VARCHAR (n)
String	TEXT(CLOB)	STRING	STRING

4.2.3.5 Importing Data

Viewing Data in the MRS Data Source by Directly Querying the Foreign Table

If the data amount is small, you can directly run SELECT to query the foreign table and view the data in the MRS data source.

Step 1 Run the following command to query data from the foreign table:

```
SELECT * FROM foreign_product_info;
```

If the query result is the same as the data in **Data File**, the import is successful. The following information is displayed at the end of the query result:

```
(20 rows)
```

After data is queried, you can insert the data to common tables in the database.

----End

Querying Data After Importing It

You can query the MRS data after importing it to GaussDB(DWS).

Step 1 Create a table in GaussDB(DWS) to store imported data.

The target table structure must be the same as the structure of the foreign table created in **Creating a Foreign Table**. That is, both tables must have the same number of columns and column types.

For example, create a table named **product_info**. The table example is as follows:

```
DROP TABLE IF EXISTS product_info;
CREATE TABLE product_info
(
    product_price                integer        ,
    product_id                   char(30)       ,
    product_time                 date           ,
    product_level                char(10)       ,
    product_name                 varchar(200)   ,
    product_type1                varchar(20)    ,
    product_type2                char(10)       ,
    product_monthly_sales_cnt    integer        ,
    product_comment_time         date           ,
```

```
    product_comment_num          integer      ,
    product_comment_content      varchar(200)
)
with (
orientation = column,
compression=middle
)
DISTRIBUTE BY HASH (product_id);
```

Step 2 Run the **INSERT INTO .. SELECT ..** command to import data from the foreign table to the target table.

Example:

```
INSERT INTO product_info SELECT * FROM foreign_product_info;
```

If information similar to the following is displayed, the data has been imported.
```
INSERT 0 20
```

Step 3 Run the following **SELECT** command to view data imported from MRS to GaussDB(DWS):
```
SELECT * FROM product_info;
```

If the query result is the same as the data in **Data File**, the import is successful. The following information is displayed at the end of the query result:

```
(20 rows)
```

----End

4.2.3.6 Deleting Resources

After completing operations in this tutorial, if you no longer need to use the resources created during the operations, you can delete them to avoid resource waste or quota occupation.

Deleting the Foreign Table and Target Table

Step 1 (Optional) If operations in **Querying Data After Importing It** have been performed, run the following command to delete the target table:
```
DROP TABLE product_info;
```

Step 2 Run the following command to delete the foreign table:
```
DROP FOREIGN TABLE foreign_product_info;
```

----End

Deleting the Manually Created Foreign Server

If operations in **Manually Creating a Foreign Server** have been performed, perform the following steps to delete the foreign server, database, and user:

Step 1 Use the client provided by GaussDB(DWS) to connect to the database where the foreign server resides as the user who created the foreign server.

You can use the **gsql** client to log in to the database in either of the following ways:

- If you have logged in to the gsql client, run the following command to switch the database and user:

\c *mydatabase dbuser*;

Enter the password as prompted.

- If you have logged in to the gsql client, you can run the **\q** command to exit gsql, and run the following command to reconnect to it:
  gsql -d *mydatabase* -h *192.168.2.30* -U *dbuser* -p 8000 -r

  Enter the password as prompted.

Step 2 Delete the manually created foreign server.

Run the following command to delete the server. For details about the syntax, see DROP SERVER.

DROP SERVER *hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca*;

The foreign server is deleted if the following information is displayed:

DROP SERVER

View the foreign server.

SELECT * FROM pg_foreign_server WHERE
srvname='*hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca*';

The server is successfully deleted if the returned result is as follows:

```
 srvname | srvowner | srvfdw | srvtype | srvversion | srvacl | srvoptions
---------+----------+--------+---------+------------+--------+------------
(0 rows)
```

Step 3 Delete the customized database.

Connect to the default database **postgres** through the database client tool provided by GaussDB(DWS).

If you have logged in to the database using the **gsql** client, run the following command to switch the database and user:

\c postgres

Enter your password as prompted.

Run the following command to delete the customized database:

DROP DATABASE *mydatabase*;

The database is deleted if the following information is displayed:

DROP DATABASE

Step 4 Delete the common user created in this example as the administrator.

Connect to the database as a database administrator through the database client tool provided by GaussDB(DWS).

If you have logged in to the database using the **gsql** client, run the following command to switch the database and user:

\c postgres dbadmin

Run the following command to reclaim the permission for creating foreign servers:
REVOKE ALL ON FOREIGN DATA WRAPPER hdfs_fdw **FROM** *dbuser*;

The name of **FOREIGN DATA WRAPPER** must be **hdfs_fdw**. **dbuser** is the username for creating **SERVER**.

Run the following command to delete the user:

DROP USER *dbuser*,

You can run the **\du** command to query for the user and check whether the user has been deleted.

----End

4.2.3.7 Error Handling

The following error information indicates that GaussDB(DWS) is to read an ORC data file but the actual file is in TXT format. Therefore, create a table of the Hive ORC type and store the data to the table.

ERROR: dn_6009_6010: Error occurs while creating an orc reader for file /user/hive/warehouse/
products_info.txt, detail can be found in dn log of dn_6009_6010.

4.2.4 Importing Data from One GaussDB(DWS) Cluster to Another

Function

You can create foreign tables to perform associated queries and import data between clusters.

Scenarios

- Import data from one GaussDB(DWS) cluster to another.
- Perform associated queries between clusters.

Precautions

- The two clusters must be in the same region and AZ, and can communicate with each other through the VPC network.
- The created foreign table must be of the same type and have the same columns as its corresponding remote table, which can only be a row-store, column-store, hash, or replication table.
- If the associated table in another cluster is a replication table or has data skew, the query performance may be poor.
- The status of the two clusters is **Normal**.
- Do not modify, add, or delete the DDL of the source data table in the remote cluster. Otherwise, the query results may be inconsistent.
- The two clusters can process SQL on other GaussDB databases based on a foreign table.
- You are advised to configure LVS. If it is not configured, you are advised to use multiple CNs as the server addresses. Do not write the CN addresses of multiple clusters together.
- Ensure that the two databases have the same encoding. Otherwise, an error may occur or the received data may be garbled characters.

- If statistics have been collected on the remote table, run **ANALYZE** on the foreign table to obtain a better execution plan.
- Only 8.0.0 and later versions are supported.

Procedure

Step 1 Create a server.

```
CREATE SERVER server_remote FOREIGN DATA WRAPPER GC_FDW OPTIONS
  (address '10.180.157.231:8000,10.180.157.130:8000' ,
 dbname 'gaussdb',
 username 'xyz',
 password 'xxxxxx'
);
```

☐ NOTE

- **server_remote** is the server name used for the foreign table.
- **address** indicates the IP addresses and port numbers of CNs in the remote cluster. If LVS is configured, you are advised to enter only one LVS address. Otherwise, you are advised to set multiple CNs as server addresses.
- **dbname** is the database name of the remote cluster.
- **username** is the username used for connecting to the remote cluster. This user cannot be a system administrator.
- **password** is the password used for logging in to the remote cluster.

Step 2 Create a foreign table.

```
CREATE FOREIGN TABLE region
(
    R_REGIONKEY INT4,
    R_NAME TEXT,
    R_COMMENT TEXT
)
SERVER
    server_remote
OPTIONS
(
    schema_name 'test',
    table_name 'region',
    encoding 'gbk'
);
```

☐ NOTE

- Foreign table columns cannot contain any constraints.
- The column names types of the foreign table must be the same as those of its corresponding remote table.
- **schema_name** specifies the schema of the foreign table corresponding to the remote cluster. If this parameter is not specified, the default schema is used.
- **table_name** specifies the name of the foreign table corresponding to the remote cluster. If this parameter is not specified, the default foreign table name is used.
- **encoding** specifies the encoding format of the remote cluster. If this parameter is not specified, the default encoding format is used.

Step 3 View the foreign table.

```
\d+ region

                Foreign table "public.region"
 Column   | Type  | Modifiers | FDW Options | Storage  | Stats target | Description
-------------+---------+-----------+-------------+----------+--------------+-------------
```

```
r_regionkey | integer |       |          | plain   |          |
r_name      | text    |       |          | extended |          |
r_comment   | text    |       |          | extended |          |
Server: server_remote
FDW Options: (schema_name 'test', table_name 'region', encoding 'gbk')
FDW permition: read only
Has OIDs: no
Distribute By: ROUND ROBIN
Location Nodes: ALL DATANODES
```

Step 4 Check the created server.

```
\des+ server_remote
                                                                List of foreign servers
   Name    | Owner | Foreign-data wrapper | Access privileges | Type | Version |
           FDW Options                                                          | Description
---------------+---------+---------------------+------------------+------+---------
+-------------------------------------------------------------
------------------------------------------------------------------------------------------------------------
+-------------
 server_remote | dbadmin | gc_fdw              |                  |      |         | (address
'10.180.157.231:8000,10.180.157.130:8000', dbname 'gaussdb'
, username 'xyz', password 'xxxxxx') |
(1 row)
```

Step 5 Use the foreign table to import data or perform associated queries.

- Import data.
  ```
  CREATE TABLE local_region
  (
      R_REGIONKEY INT4,
      R_NAME TEXT,
      R_COMMENT TEXT
  );
  INSERT INTO local_region SELECT * FROM region;
  ```

  📖 **NOTE**

  – If a connection failure is reported, check the server information and ensure that the specified clusters are connected.

  – If an error is reported, indicating that the table does not exist, check whether the **option** information of the foreign table is correct.

  – If a column mismatch error is reported, check whether the column information of the foreign table is consistent with that of the corresponding table in the remote cluster.

  – If a version inconsistency error is reported, upgrade the cluster and try again.

  – If garbled characters are displayed, check the encoding format of the source data, re-create a foreign table, and specify the correct coding format.

- Perform an associated query.
  ```
  SELECT * FROM region, local_region WHERE local_region.R_NAME = region.R_NAME;
  ```

  📖 **NOTE**

  – A foreign table can be used as a local table to perform complex jobs.

  – If statistics have been collected on the remote cluster, run **ANALYZE** on the foreign table to obtain a better execution plan.

  – If there are fewer DNs in the local cluster than in the remote cluster, the local cluster needs to use SMP for better performance.

Step 6 Delete the foreign table.

DROP FOREIGN TABLE region;

----End

4.2.5 GDS-based Cross-Cluster Interconnection

Function

With data processing based on foreign tables, GDS is used to transfer data and synchronize data between multiple clusters.

Scenarios

Data is synchronized from one cluster to another. Full data synchronization and data synchronization based on filter criteria are supported.

Important Notes

- The column name and type of the created foreign table must be the same as those of the corresponding source table, and the tables in the remote cluster must be row-store or column-store.

- Before running the synchronization statement, ensure that the tables to be synchronized exist in the local and remote clusters.

- The status of the two clusters is **Normal**.

- Both clusters must be able to connect to each other using GDS.

- The database code of both clusters must be the same. Otherwise, an error may be reported or the received data may be garbled characters.

- The compatible database types specified for both clusters must be the same. Otherwise, an error may be reported or the received data may be garbled characters.

- Ensure that the user performing table synchronization has the permission to access those tables.

- Foreign tables for interconnection can be used only for cross-cluster data synchronization. In other scenarios, errors may occur or the operation may be invalid.

- The foreign tables for interconnection do not support complex column expressions or complex syntax, including **join**, **sort**, **cursor**, **with**, and **set**.

- SQL statements that are not pushed down cannot use this feature to synchronize data.

- The explain plan and logical cluster are not supported.

- If data is synchronized from the local cluster to a remote cluster, only internal table query is supported.

- The GDS specified by the **syncsrv** option of foreign servers does not support the SSL mode.

- After data synchronization is complete, only the number of data rows is verified.

- The maximum number of concurrent services cannot be greater than half of the value of the GDS startup parameter **-t** and cannot be greater than the value of **max_active_statements**. Otherwise, services may fail due to timeout.

Preparations

- Configure the interconnection between both clusters.
- Plan and deploy GDS servers and ensure that all GDS servers can communicate with all nodes in both clusters. For details about how to deploy GDS, see **Installing, Configuring, and Starting GDS**.

Procedure

Assume that the table **tbl_remote** in the remote cluster is to be synchronized with the table **tbl_local** in the local cluster and the user performing the synchronization is **user_remote**. Note that the user must have the permission to access the **tbl_remote** table.

Step 1 Create a server.

```
CREATE SERVER server_remote FOREIGN DATA WRAPPER GC_FDW OPTIONS(
  address '192.168.178.207:8109',
  dbname 'db_remote',
  username 'user_remote',
  password 'xxxxxxxx',
  syncsrv 'gsfs://192.168.178.129:8789|gsfs://192.168.178.129:8790'
);
```

- **server_remote** indicates the server name, which is used by the foreign table for interconnection.
- **address** indicates the IP address and port number of the CN in the remote cluster. Only one address is allowed.
- **dbname** indicates the database name of the remote cluster.
- **username** indicates the username used for connecting to the remote cluster. This user cannot be a system administrator.
- **password** indicates the password used for connecting to the remote cluster.
- **syncsrv** indicates the IP address and port number of the GDS server. If there are multiple addresses, use vertical bars (|) to separate them. The function of **syncsrv** is similar to that of **location** in the GDS foreign table.

  📖 **NOTE**

  GaussDB(DWS) tests the network connected to the GDS addresses set by **syncsrv**.
  - The test can only show the network status between the local cluster and the GDSs, but cannot show the network status between the remote cluster and GDS. You need to check the error message.
  - After removing the unavailable GDSs, select a proper number of GDSs that do not cause service suspension to synchronize data.

Step 2 Create a foreign table for interconnection.

```
CREATE FOREIGN TABLE ft_tbl(
  col_1    type_name,
  col_2    type_name,
  …
) SERVER server_remote OPTIONS (
  schema_name 'schema_remote',
  table_name 'tbl_remote',
  encoding 'utf8'
);
```

- **schema_name** indicates the schema that the remote cluster table belongs to. If this option is not specified, **schema_name** is set to the schema of the foreign table.

- **table_name** indicates the remote cluster table name. If this option is not specified, **table_name** is set to the name of the foreign table.
- **encoding** indicates the encoding format of the remote cluster. If this option is not specified, the default encoding format of the source cluster database is used.

📖 NOTE

- The values of **schema_name** and **table_name** are case sensitive and must be the same as those of the remote schema and table.
- The foreign table for interconnection cannot contain any constraints in its columns.
- The column names and column types of the foreign table must be the same as those of the **tbl_remote** table.
- **SERVER** must be set to the server created in **Step 1** and must contain the **syncsrv** attribute.

Step 3 Use the foreign table for interconnection to synchronize data.

- If the local cluster is the destination cluster, you can run the following statements:

  Full data synchronization of all columns:

  INSERT INTO tbl_local SELECT * FROM ft_tbl;

  Data synchronization of all columns based on filter criteria:

  INSERT INTO tbl_local SELECT * FROM ft_tbl WHERE col_2 = XX;

  Full data synchronization of some columns:

  INSERT INTO tbl_local (col_1) SELECT col_1 FROM ft_tbl;

  Data synchronization of some columns based on filter criteria:

  INSERT INTO tbl_local (col_1) SELECT col_1 FROM ft_tbl WHERE col_2 = XX;

- If the local cluster is the source cluster, you can run the following statements:

  Synchronization of unsharded tables:

  INSERT INTO ft_tbl SELECT * FROM tbl_local;

  Data synchronization of the **join** results:

  INSERT INTO ft_tbl SELECT * FROM tbl_local1 join tbl_local2 ON XXX;

📖 NOTE

- If a connection failure is reported, check the server information and ensure that both clusters are connected.
- If an error indicating GDS connection failure is reported, check whether the GDS server specified by **syncsrv** has been started and whether it can communicate with all nodes in both clusters.
- If an error is reported indicating that the table does not exist, check whether the **option** information of the foreign table is correct.
- If an error is reported indicating that the column does not exist, check whether the column name of the foreign table is the same as that of the source table.
- If an error message is displayed indicating that a column is repeatedly defined, check whether the column name is too long. If yes, use the AS alias to simplify the column name.
- If an error is reported indicating that the column type cannot be parsed, check whether the statement contains a column expression.
- If a column mismatch error is reported, check whether the column information of the foreign table is the same as that of the corresponding table in the remote cluster.
- If an error is reported indicating that the syntax is not supported, check whether complex syntax is used, such as **join**, **distinct**, and **sort** .
- If garbled characters are displayed, check whether the encoding formats of both databases are the same.
- If the local cluster is the source cluster, there is a low probability that data is successfully synchronized to the remote cluster but the local cluster returns an execution failure. In this case, you are advised to check the number of synchronized data records.
- If the local cluster is the source cluster, data synchronization controlled by transaction blocks and sub-transactions can be queried only after the total transaction is committed.

Step 4 Delete the foreign table for interconnection.
```
DROP FOREIGN TABLE ft_tbl;
```

----End

4.2.6 Using a gsql Meta-Command to Import Data

The **gsql** tool of GaussDB(DWS) provides the **\copy** meta-command to import data.

\copy Command

For details about the **\copy** command, see **Table 4-12**.

Table 4-12 \copy meta-command

Syntax	Description
\copy { table [(column_list)] | (query) } { from | to } { filename | stdin | stdout | pstdin | pstdout } [with] [binary] [oids] [delimiter [as] 'character'] [null [as] 'string'] [csv [header] [quote [as] 'character'] [escape [as] 'character'] [force quote column_list | *] [force not null column_list]]	You can run this command to import or export data after logging in to the database on any gsql client. Different from the **COPY** statement in SQL, this command performs read/write operations on local files rather than files on database servers. The accessibility and permissions of the local files are restricted to local users. **NOTE** **\copy** only applies to small-batch data import with uniform formats but poor error tolerance capability. GDS or **COPY** is preferred for data import.

Parameter Description

- table

  Specifies the name (possibly schema-qualified) of an existing table.

  Value range: an existing table name

- column_list

  Specifies an optional list of columns to be copied.

  Value range: any field in the table. If the column list is not specified, all columns in the table will be copied.

- query

  Specifies that the results will be copied.

  Valid value: a **SELECT** or **VALUES** command in parentheses.

- filename

  Specifies the absolute path of a file. To run the **\copy** command, the user must have the write permission for this path.

- stdin

  Specifies that input comes from the client application.

- stdout

  Specifies that output goes to the client application.

- pstdin

  Specifies that input comes from the gsql client.

- pstout

- Specifies that output goes to the gsql client.

- binary

  Specifies that data is stored and read in binary mode instead of text mode. In binary mode, you cannot declare **DELIMITER**, **NULL**, or **CSV**. After specifying BINARY, CSV, FIXED and TEXT cannot be specified through **option** or **copy_option**.

- oid

  Specifies the internal OID to be copied for each row.

  > **NOTE**
  >
  > An error is raised if OIDs are specified for a table that does not have OIDs, or in the case of copying a query.

  Valid value: **true**, **on**, **false**, and **off**.

  Default value: **false**

- delimiter [as] 'character'

  Specifies the character that separates columns within each row (line) of the file.

  > **NOTE**
  >
  > - A delimiter cannot be **\r** or **\n**.
  > - A delimiter cannot be the same as the **null** value. The delimiter of CSV data cannot be same as the **quote** value.
  > - The delimiter of TEXT data cannot contain any of the following characters: \.abcdefghijklmnopqrstuvwxyz0123456789
  > - The data length of a single row should be less than 1 GB. A row that has many columns using long delimiters cannot contain much valid data.
  > - You are advised to use multi-characters and invisible characters for delimiters. For example, you can use the multiple-character delimiter "$^&" and invisible delimiters, such as E'\x07', E'\x08', and E'\x1b'.

  Value range: a multi-character delimiter within 10 bytes.

  Default value:

  – A tab character in TEXT format

  – A comma (,) in CSV format

  – No delimiter in FIXED format

- null [as] 'string'

  Specifies that a string represents a null value in a data file.

  Value range:

  – A null value cannot be **\r** or **\n**. The maximum length is 100 characters.

  – A null value cannot be the same as the **delimiter** or **quote** value.

  Default value:

  – An empty string without quotation marks in CSV format

  – **\N** in TEXT format

- header

  Specifies whether a data file contains a table header. **header** is available only for CSV and FIXED files.

  In data import scenarios, if **header** is **on**, the first row of the data file will be identified as the header and ignored. If **header** is **off**, the first row will be identified as a data row.

  If header is **on**, **fileheader** must be specified. **fileheader** specifies the content in the header. If header is **off**, the exported file does not contain a header.

  Valid value: **true**, **on**, **false**, and **off**.

  Default value: **false**

- quote [as] 'character'

  Specifies the quote character for a CSV file.

  Default value: double quotation mark ("").

  📖 NOTE

  - The **quote** value cannot be the same as the **delimiter** or **null** value.
  - The **quote** value must be a single-byte character.
  - You are advised to use invisible characters as quotes, for example, E'\x07', E'\x08', and E'\x1b'.

- escape [as] 'character'

  This option is allowed only when using CSV format. This must be a single one-byte character.

  Default value: double quotation mark (""). If the value is the same as the **quote** value, it will be replaced with **\0**.

- force quote column_list | *

  In CSV COPY TO mode, forces quoting to be used for all not-null values in each specified column. NULL will not be quoted.

  Value range: an existing column.

- force not null column_list

  In CSV COPY FROM mode, processes each specified column as though it were quoted and hence not a null value.

  Value range: an existing column.

Examples

Create the target table **copy_example**.

```
create table copy_example
(
    col_1 integer,
    col_2 text,
    col_3 varchar(12),
    col_4 date,
    col_5 time
);
```

- Example 1: Copy data from stdin to the target table **copy_example**.

  ```
  \copy copy_example from stdin csv;
  ```

  When the **>>** characters are displayed, enter data. Enter a backslash and a period (\.) to end your input.

  ```
  Enter data to be copied followed by a newline.
  End with a backslash and a period on a line by itself.
  >> 1,"iamtext","iamvarchar",2006-07-07,12:00:00
  >> \.
  ```

- Example 2: The **example.csv** file is in the local directory **/local/data/** and the file contains the header line. (|) is used as the delimiter, and the double quotation marks are used for **quote**. The content is as follows:

  iamheader

  1|"iamtext"|"iamvarchar"|2006-07-07|12:00:00

  2|"iamtext"|"iamvarchar"|2022-07-07|19:00:02

Import data from the local file **example.csv** to the target table **copy_example**. If the header option is **on**, the first row is automatically ignored. By default, quotation marks are used for **quote**.

```
\copy copy_example from  '/local/data/example.csv' with(header 'on', format 'csv', delimiter '|',
date_format 'yyyy-mm-dd',  time_format 'hh24:mi:ss');
```

- Example 3: The **example.csv** file is in the local directory **/local/data/**. The comma (,) is used as the delimiter, and the quotation mark (") is used for **quote**. The last field is missing in the first line, and one more field is added in the second line. The content is as follows:

  1,"iamtext","iamvarchar",2006-07-07

  2,"iamtext","iamvarchar",2022-07-07,19:00:02,12:00:00

  Import data from the local file **example.csv** to the target table **copy_example**. The default delimiter is (,). Therefore, you do not need to specify the delimiter. Because the fault tolerance parameters **IGNORE_EXTRA_DATA** and **FILL_MISSING_FIELD** are specified, the missing fields will be replaced with **NULL**, the extra fields are ignored.

  ```
  \copy copy_example from '/local/data/example.csv' with( format 'csv', date_format 'yyyy-mm-dd',
  time_format 'hh24:mi:ss', IGNORE_EXTRA_DATA 'true', FILL_MISSING_FIELD 'true');
  ```

- Example 4: Export the content of the **copy_example** table to stdout in CSV format, use double quotation marks as for **quote**, and use quotes to enclose the fourth and fifth columns.
  ```
  \copy copy_example to stdout CSV quote as '"' force quote col_4,col_5;
  ```

4.2.7 Running the COPY FROM STDIN Statement to Import Data

4.2.7.1 Data Import Using COPY FROM STDIN

This method is applicable to low-concurrency scenarios where a small volume of data is to be imported.

Use either of the following methods to write data to GaussDB(DWS) using the **COPY FROM STDIN** statement:

- Write data into GaussDB(DWS) by typing.
- Import data from a file or database to GaussDB(DWS) through the CopyManager interface driven by JDBC. You can use any parameters in the **COPY** syntax.

4.2.7.2 Introduction to the CopyManager Class

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 is inherited from the java.lang.Object class. The declaration of the class is as follows:

```
public class CopyManager
extends Object
```

Constructor Method

public CopyManager(BaseConnection connection)

throws SQLException

Basic Methods

Table 4-13 Basic methods of CopyManager

Return Value	Method	Description	Throws
CopyIn	copyIn(String sql)	-	SQLException
long	copyIn(String sql, InputStream from)	Uses **COPY FROM STDIN** to quickly import data to tables in a database from InputStream.	SQLException,IOEx.xception
long	copyIn(String sql, InputStream from, int bufferSize)	Uses **COPY FROM STDIN** to quickly import data to tables in a database from InputStream.	SQLException,IOException
long	copyIn(String sql, Reader from)	Uses **COPY FROM STDIN** to quickly import data to tables in a database from Reader.	SQLException,IOException
long	copyIn(String sql, Reader from, int bufferSize)	Uses **COPY FROM STDIN** to quickly import data to tables in a database from Reader.	SQLException,IOException
CopyOut	copyOut(String sql)	-	SQLException
long	copyOut(String sql, OutputStream to)	Sends the result set of **COPY TO STDOUT** from a database to the OutputStream class.	SQLException,IOException

Return Value	Method	Description	Throws
long	copyOut(String sql, Writer to)	Sends the result set of **COPY TO STDOUT** from a database to the Writer class.	SQLException,IOException

4.2.7.3 Example: 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);
        }

     //Export the query result of SELECT * FROM migration_table to the local file d:/data.txt.
     try {
     copyToFile(conn, "d:/data.txt", "(SELECT * FROM migration_table)");
    } catch (SQLException e) {
// TODO Auto-generated catch block
e.printStackTrace();
    } catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
    }
```

```
    //Import data from the d:/data.txt file to the migration_table_1 table.
    try {
    copyFromFile(conn, "d:/data.txt", tablename1);
 } catch (SQLException e) {
// TODO Auto-generated catch block
        e.printStackTrace();
} catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}

    //Export the data from the migration_table_1 table to the d:/data1.txt file.
    try {
    copyToFile(conn, "d:/data1.txt", tablename1);
 } catch (SQLException e) {
// TODO Auto-generated catch block
e.printStackTrace();
 } catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
    }

public static void copyFromFile(Connection connection, String filePath, String tableName)
        throws SQLException, IOException {

    FileInputStream fileInputStream = null;

    try {
        CopyManager copyManager = new CopyManager((BaseConnection)connection);
        fileInputStream = new FileInputStream(filePath);
        copyManager.copyIn("COPY " + tableName + " FROM STDIN", fileInputStream);
    } finally {
        if (fileInputStream != null) {
            try {
                fileInputStream.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}

public static void copyToFile(Connection connection, String filePath, String tableOrQuery)
        throws SQLException, IOException {

    FileOutputStream fileOutputStream = null;

    try {
        CopyManager copyManager = new CopyManager((BaseConnection)connection);
        fileOutputStream = new FileOutputStream(filePath);
        copyManager.copyOut("COPY " + tableOrQuery + " TO STDOUT", fileOutputStream);
    } finally {
        if (fileOutputStream != null) {
            try {
                fileOutputStream.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
}
```

4.2.7.4 Example: 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**.

```java
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");          //DWS username
        String pass = new String(" ********");     //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;
    }
}
```

4.3 Full Database Migration

4.3.1 Using DRS to Import Data to GaussDB(DWS)

You can use Data Replication Service (DRS) to import data from other data sources to databases of a GaussDB(DWS) cluster. Currently, the following data sources can be imported:

- MySQL
- DDM
- PostgreSQL (OBT)
- Oracle (OBT)
- GaussDB Distributed (OBT)

For details, see **DRS Real-Time Synchronization**.

📖 NOTE

The data sources of PostgreSQL, Oracle, and GaussDB Distributed are in the open beta test (OBT) phase. You can apply for the OBT by creating a service ticket on the DRS management console.

4.3.2 Using CDM to Migrate Data to GaussDB(DWS)

You can use CDM to migrate data from other data sources (for example, MySQL) to the databases in clusters on GaussDB(DWS).

For details about scenarios where CDM is used to migrate data to GaussDB(DWS), see the following sections of *Cloud Data Migration User Guide*:

- **Getting Started**: describes how to use CDM to migrate local MySQL databases to GaussDB(DWS).

4.3.3 Using DSC to Migrate SQL Scripts

The DSC is a CLI tool running on the Linux or Windows OS. It is dedicated to providing customers with simple, fast, and reliable application SQL script

migration services. It parses the SQL scripts of source database applications using the built-in syntax migration logic, and converts them to SQL scripts applicable to GaussDB(DWS) databases. You do not need to connect the DSC to a database. It can migrate data in offline mode without service interruption. In GaussDB(DWS), you can run the migrated SQL scripts to restore the database, thereby easily migrating offline databases to the cloud.

The DSC can migrate SQL scripts of Teradata, Oracle, Netezza, MySQL, and DB2 databases.

Downloading the DSC SQL Migration Tool

Step 1 Log in to the GaussDB(DWS) management console.

Step 2 In the navigation tree on the left, click **Connection Management**.

Step 3 In the **Download Client and Driver** area, click **here** to download the DSC migration tool.

If you have clusters of different versions, the system displays a dialog box, prompting you to select the cluster version and download the client corresponding to the cluster version. In the cluster list on the **Cluster Management** page, click the name of the specified cluster and click the **Basic Information** tab to view the cluster version.

Figure 4-8 Downloading the tool
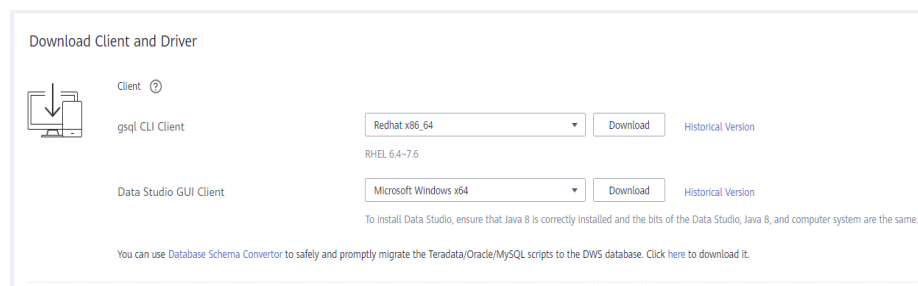
Step 4 After downloading the DSC tool to the local PC, use WinSCP to upload it to a Linux host.

The user who uploads the tool must have the full control permission on the target directory of the Linux host.

----End

Operation Guide for the DSC SQL Syntax Migration Tool

For details, see **DSC - SQL Syntax Migration Tool**.

4.4 Metadata Migration

4.4.1 Using gs_dump and gs_dumpall to Export Metadata

4.4.1.1 Overview

GaussDB(DWS) provides gs_dump and gs_dumpall to export required database objects and related information. To migrate database information, you can use a tool to import the exported metadata to a target database. gs_dump exports a single database or its objects. gs_dumpall exports all databases or global objects in a cluster. For details, see **Table 4-14**.

Table 4-14 Application scenarios

Applica tion Scenari o	Export Granularity	Export Format	Import Method
Exportin g a single databas e	**Database-level export** ● Export full information of a database. You can use the exported information to create a same database containing the same data as the current one. ● Export all object definitions of a database, including the definitions of the database, functions, schemas, tables, indexes, and stored procedures. You can use the exported object definitions to quickly create a same database as the current one, without data. ● Export data of a database.	● Plain text ● Custom ● Director y ● .tar	● For details about how to import data files in text format, see **Using a gsql Meta-Command to Import Data**. ● For details about how to import data files in .tar, directory, or custom format, see **Using gs_restore to Import Data**.
	Schema-level export ● Export full information of a schema. ● Export data of a schema. ● Export all object definitions of a schema, including the definitions of tables, stored procedures, and indexes.		
	Table-level export ● Export full information of a table. ● Export data of a table. ● Export the definition of a table.		

Application Scenario	Export Granularity	Export Format	Import Method
Exporting all databases in a cluster	**Database-level export** ● Export full information of a cluster. You can use the exported information to create a same cluster containing the same databases, global objects, and data as the current one. ● Export all object definitions of a cluster, including the definitions of tablespaces, databases, functions, schemas, tables, indexes, and stored procedures. You can use the exported object definitions to quickly create a same cluster as the current one, containing the same databases and tablespaces but without data. ● Export data of a cluster.	Plain text	For details about how to import data files, see **Using a gsql Meta-Command to Import Data**.
	Global object export ● Export tablespaces. ● Export roles. ● Export tablespaces and roles.		

gs_dump and gs_dumpall use **-U** to specify the user that performs the export. If the specified user does not have the required permission, data cannot be exported. In this case, you can set **--role** in the export command to the role that has the permission. Then, gs_dump or gs_dumpall uses the specified role to export data. See **Table 4-14** for application scenarios and **Data Export By a User Without Required Permissions** for operation details.

gs_dump and gs_dumpall encrypt the exported data files. These files are decrypted before being imported to prevent data disclosure for higher database security.

When gs_dump or gs_dumpall is used to export data from a cluster, other users can still access (read data from and write data to) databases in the cluster.

gs_dump and gs_dumpall can export complete, consistent data. For example, if gs_dump is used to export database A or gs_dumpall is used to export all databases from a cluster at T1, data of database A or all databases in the cluster at that time point will be exported, and modifications on the databases after that time point will not be exported.

Obtain gs_dump and gs_dumpall by decompressing the **gsql CLI client** package.

Precautions

- Do not modify an exported file or its content. Otherwise, restoration may fail.
- For data consistency and integrity, gs_dump and gs_dumpall set a share lock for a table to be dumped. If a share lock has been set for the table in other transactions, gs_dump and gs_dumpall lock the table after the lock is released. If the table cannot be locked within the specified time, the dump fails. You can customize the timeout duration to wait for lock release by specifying the **--lock-wait-timeout** parameter.
- During an export, gs_dumpall reads all tables in a database. Therefore, you need to connect to the database as a cluster administrator to export a complete file. When you use gsql to import scripts, cluster administrator permissions are also required to add users and user groups, and create databases.
- By default, the definitions of all views in the GaussDB(DWS) database contain the prefix of table names or aliases (in **tab.col** format). Therefore, the definitions may be inconsistent with the original ones. As a result, the base table corresponding to the rebuilt view column is incorrect and an error is reported. However, this rarely happens. To prevent this problem, you are advised to set the GUC parameter **behavior_compat_options** to **compat_display_ref_table** when exporting view definitions, so the exported definitions are consistent with the original statements.

4.4.1.2 Exporting a Single Database

4.4.1.2.1 Exporting a Database

You can use gs_dump to export data and all object definitions of a database from GaussDB(DWS). You can specify the information to be exported as follows:

- Export full information of a database, including its data and all object definitions.

  You can use the exported information to create a same database containing the same data as the current one.

- Export all object definitions of a database, including the definitions of the database, functions, schemas, tables, indexes, and stored procedures.

  You can use the exported object definitions to quickly create a same database as the current one, without data.

- Export data of a database.

Procedure

Step 1 **Preparing an ECS as the gsql Client Host**.

Step 2 Download the gsql client and use an SSH transfer tool (such as WinSCP) to upload it to the Linux server where gsql is to be installed. For details, see **Downloading the Client**.

The user who uploads the client must have the full control permission on the target directory on the host to which the client is uploaded.

Alternatively, you can remotely log in to the Linux host where the gsql is to be installed in SSH mode and run the following command in the Linux command window to download the gsql client:

```
wget https://obs.myhuaweicloud.com/dws/download/dws_client_8.x.x_redhat_x64.zip --no-check-certificate
```

Step 3 Run the following commands to decompress the client:

```
cd <Path_for_storing_the_client>
unzip dws_client_8.x.x_redhat_x64.zip
```

Where,

- *<Path_for_storing_the_client>*: Replace it with the actual path.

- *dws_client_8.1.x_redhat_x86.zip*: This is the client tool package of **RedHat x86**. Replace it with the actual one.

Step 4 Run the following command to configure the GaussDB(DWS) client:

```
source gsql_env.sh
```

If the following information is displayed, the GaussDB(DWS) client is successfully configured:

```
All things done.
```

Step 5 Use gs_dump to export data of the database **gaussdb**.

gs_dump -W *password* **-U** *jack* **-f** */home//backup/postgres_backup.tar* **-p 8000** gaussdb *-h 10.10.10.100* **-F** *t*

Table 4-15 Common parameters

Parameter	Description	Example Value
-U	Username for connecting to the database. If this parameter is not configured, the username of the connected database is used.	-U jack
-W	User password for database connection. ● This parameter is not required for database administrators if the trust policy is used for authentication. ● If you connect to the database without specifying this parameter and you are not a database administrator, you will be prompted to enter the password.	-W *Password*
-f	Folder to store exported files. If this parameter is not specified, the exported files are stored in the standard output.	-f /home//backup/ *postgres*_backup.tar

Parameter	Description	Example Value
-p	Name extension of the TCP port on which the server is listening or the local Unix domain socket. This parameter is configured to ensure connections.	-p 8000
-h	*Cluster address*: If a public network address is used for connection, set this parameter to **Public Network Address** or **Public Network Domain Name**. If a private network address is used for connection, set this parameter to **Private Network Address** or **Private Network Domain Name**.	-h 10.10.10.100
dbname	Name of the database to be exported.	gaussdb
-F	Format of exported files. The values of **-F** are as follows: ● **p**: plain text ● **c**: custom ● **d**: directory ● **t**: .tar	-F t

For details about other parameters, see "gs_dump" in the *Tool Guide*.

----End

Examples

Example 1: Use gs_dump to run the following command to export full information of the database **gaussdb** and compress the exported files in SQL format.

```
gs_dump -W password -U jack -f /home//backup/postgres_backup.sql -p 8000  -h 10.10.10.100 gaussdb -Z
8 -F p
gs_dump[port=''][gaussdb][2017-07-21 15:36:13]: dump database gaussdb successfully
gs_dump[port=''][gaussdb][2017-07-21 15:36:13]: total time: 3793  ms
```

Example 2: Use gs_dump to run the following command to export data of the database **gaussdb**, excluding object definitions. The exported files are in a custom format.

```
gs_dump -W Password -U jack -f /home//backup/postgres_data_backup.dmp -p 8000 -h 10.10.10.100
gaussdb -a -F c
gs_dump[port=''][gaussdb][2017-07-21 15:36:13]: dump database gaussdb successfully
gs_dump[port=''][gaussdb][2017-07-21 15:36:13]: total time: 3793  ms
```

Example 3: Use gs_dump to run the following command to export object definitions of the database **gaussdb**. The exported files are in SQL format.

```
--Before the export, the nation table contains data.
select n_nationkey,n_name,n_regionkey from nation limit 3;
 n_nationkey |         n_name          | n_regionkey
-------------+-------------------------+-------------
           0 | ALGERIA                 |           0
           3 | CANADA                  |           1
          11 | IRAQ                    |           4
(3 rows)

gs_dump -W password -U jack -f /home//backup/postgres_def_backup.sql -p 8000 -h 10.10.10.100 gaussdb -
s -F p
gs_dump[port=''][gaussdb][2017-07-20 15:04:14]: dump database gaussdb successfully
gs_dump[port=''][gaussdb][2017-07-20 15:04:14]: total time: 472 ms
```

Example 4: Use gs_dump to run the following command to export object definitions of the database **gaussdb**. The exported files are in text format and are encrypted.

```
gs_dump -W password -U jack -f /home//backup/postgres_def_backup.sql -p 8000 -h 10.10.10.100 gaussdb
--with-encryption AES128 --with-key 1234567812345678 -s -F p
gs_dump[port=''][gaussdb][2018-11-14 11:25:18]: dump database gaussdb successfully
gs_dump[port=''][gaussdb][2018-11-14 11:25:18]: total time: 1161  ms
```

4.4.1.2.2 Exporting a Schema

You can use gs_dump to export data and all object definitions of a schema from GaussDB(DWS). You can export one or more specified schemas as needed. You can specify the information to be exported as follows:

- Export full information of a schema, including its data and object definitions.
- Export data of a schema, excluding its object definitions.
- Export the object definitions of a schema, including the definitions of tables, stored procedures, and indexes.

Procedure

Step 1 **Preparing an ECS as the gsql Client Host**.

Step 2 Download the gsql client and use an SSH transfer tool (such as WinSCP) to upload it to the Linux server where gsql is to be installed. For details, see **Downloading the Client**.

The user who uploads the client must have the full control permission on the target directory on the host to which the client is uploaded.

Alternatively, you can remotely log in to the Linux host where the gsql is to be installed in SSH mode and run the following command in the Linux command window to download the gsql client:

```
wget https://obs.myhuaweicloud.com/dws/download/dws_client_8.x.x_redhat_x64.zip --no-check-certificate
```

Step 3 Run the following commands to decompress the client:

```
cd <Path_for_storing_the_client>
unzip dws_client_8.x.x_redhat_x64.zip
```

Where,

- *<Path_for_storing_the_client>*: Replace it with the actual path.
- *dws_client_8.1.x_redhat_x86.zip*: This is the client tool package of **RedHat x86**. Replace it with the actual one.

Step 4 Run the following command to configure the GaussDB(DWS) client:

```
source gsql_env.sh
```

If the following information is displayed, the GaussDB(DWS) client is successfully configured:

```
All things done.
```

Step 5 Use gs_dump to run the following command to export the **hr** and **public** schemas.

```
gs_dump -W Password -U jack -f /home//backup/MPPDB_schema_backup -p 8000 -h 10.10.10.100
human_resource -n hr -F d
```

Table 4-16 Common parameters

Parameter	Description	Example Value
-U	Username for connecting to the database. If this parameter is not configured, the username of the connected database is used.	-U jack
-W	User password for database connection. • This parameter is not required for database administrators if the trust policy is used for authentication. • If you connect to the database without specifying this parameter and you are not a database administrator, you will be prompted to enter the password.	-W Password
-f	Folder to store exported files. If this parameter is not specified, the exported files are stored in the standard output.	-f /home//backup/ MPPDB_schema_backu p
-p	Name extension of the TCP port on which the server is listening or the local Unix domain socket. This parameter is configured to ensure connections.	-p 8000
-h	*Cluster address*: If a public network address is used for connection, set this parameter to **Public Network Address** or **Public Network Domain Name**. If a private network address is used for connection, set this parameter to **Private Network Address** or **Private Network Domain Name**.	-h 10.10.10.100

Parameter	Description	Example Value
dbname	Name of the database to be exported.	human_resource
-n	Names of schemas to be exported. Data of the specified schemas will also be exported. ● Single schema: Enter **-n** *schemaname*. ● Multiple schemas: Enter **-n** *schemaname* for each schema.	● Single schema: **-n hr** ● Multiple schemas: **-n hr -n public**
-F	Format of exported files. The values of **-F** are as follows: ● **p**: plain text ● **c**: custom ● **d**: directory ● **t**: .tar	-F d

For details about other parameters, see "gs_dump" in the *Tool Guide*.

----End

Examples

Example 1: Use gs_dump to run the following command to export full information of the **hr** schema. The exported files are compressed and stored in text format.

```
gs_dump -W password -U jack -f /home//backup/MPPDB_schema_backup.sql -p 8000 -h 10.10.10.100
human_resource -n hr -Z 6 -F p
gs_dump[port=''][human_resource][2017-07-21 16:05:55]: dump database human_resource successfully
gs_dump[port=''][human_resource][2017-07-21 16:05:55]: total time: 2425  ms
```

Example 2: Use gs_dump to run the following command to export data of the **hr** schema. The exported files are in .tar format.

```
gs_dump -W password -U jack -f /home//backup/MPPDB_schema_data_backup.tar -p 8000 -h 10.10.10.100
human_resource -n hr -a -F t
gs_dump[port=''][human_resource][2018-11-14 15:07:16]: dump database human_resource successfully
gs_dump[port=''][human_resource][2018-11-14 15:07:16]: total time: 1865  ms
```

Example 3: Use gs_dump to run the following command to export the definition of the **hr** schema. The exported files are stored in a directory.

```
gs_dump -W password -U jack -f /home//backup/MPPDB_schema_def_backup -p 8000 -h 10.10.10.100
human_resource -n hr -s -F d
gs_dump[port=''][human_resource][2018-11-14 15:11:34]: dump database human_resource successfully
gs_dump[port=''][human_resource][2018-11-14 15:11:34]: total time: 1652  ms
```

Example 4: Use gs_dump to run the following command to export the **human_resource** database excluding the **hr** schema. The exported files are in a custom format.

```
gs_dump -W password -U jack -f /home//backup/MPPDB_schema_backup.dmp -p 8000 -h 10.10.10.100
human_resource -N hr -F c
gs_dump[port=''][human_resource][2017-07-21 16:06:31]: dump database human_resource successfully
gs_dump[port=''][human_resource][2017-07-21 16:06:31]: total time: 2522  ms
```

Example 5: Use gs_dump to run the following command to export the object definitions of the **hr** and **public** schemas, encrypt the exported files, and store them in .tar format.

```
gs_dump -W password -U jack -f /home//backup/MPPDB_schema_backup1.tar -p 8000 -h 10.10.10.100
human_resource -n hr -n public -s --with-encryption AES128 --with-key 1234567812345678 -F t
gs_dump[port=''][human_resource][2017-07-21 16:07:16]: dump database human_resource successfully
gs_dump[port=''][human_resource][2017-07-21 16:07:16]: total time: 2132  ms
```

Example 6: Use gs_dump to run the following command to export the **human_resource** database excluding the **hr** and **public** schemas. The exported files are in a custom format.

```
gs_dump -W password -U jack -f /home//backup/MPPDB_schema_backup2.dmp -p 8000 -h 10.10.10.100
human_resource -N hr -N public -F c
gs_dump[port=''][human_resource][2017-07-21 16:07:55]: dump database human_resource successfully
gs_dump[port=''][human_resource][2017-07-21 16:07:55]: total time: 2296  ms
```

Example 7: Use gs_dump to run the following command to export all tables, including views, sequences, and foreign tables, in the **public** schema, and the **staffs** table in the **hr** schema, including data and table definition. The exported files are in a custom format.

```
gs_dump -W password -U jack -f /home//backup/MPPDB_backup3.dmp -p 8000 -h 10.10.10.100
human_resource -t public.* -t hr.staffs -F c
gs_dump[port=''][human_resource][2018-12-13 09:40:24]: dump database human_resource successfully
gs_dump[port=''][human_resource][2018-12-13 09:40:24]: total time: 896  ms
```

4.4.1.2.3 Exporting a Table

You can use gs_dump to export data and all object definitions of a table-level object from GaussDB(DWS). Views, sequences, and foreign tables are special tables. You can export one or more specified tables as needed. You can specify the information to be exported as follows:

- Export full information of a table, including its data and definition.
- Export data of a table.
- Export the definition of a table.

Procedure

Step 1 **Preparing an ECS as the gsql Client Host**.

Step 2 Download the gsql client and use an SSH transfer tool (such as WinSCP) to upload it to the Linux server where gsql is to be installed. For details, see **Downloading the Client**.

The user who uploads the client must have the full control permission on the target directory on the host to which the client is uploaded.

Alternatively, you can remotely log in to the Linux host where the gsql is to be installed in SSH mode and run the following command in the Linux command window to download the gsql client:

```
wget https://obs.myhuaweicloud.com/dws/download/dws_client_8.x.x_redhat_x64.zip --no-check-certificate
```

Step 3 Run the following commands to decompress the client:

```
cd <Path_for_storing_the_client>
unzip dws_client_8.x.x_redhat_x64.zip
```

Where,

- *<Path_for_storing_the_client>*: Replace it with the actual path.
- *dws_client_8.1.x_redhat_x86.zip*: This is the client tool package of **RedHat x86**. Replace it with the actual one.

Step 4 Run the following command to configure the GaussDB(DWS) client:

```
source gsql_env.sh
```

If the following information is displayed, the GaussDB(DWS) client is successfully configured:

```
All things done.
```

Step 5 Use gs_dump to run the following command to export the **hr.staffs** and **hr.employments** tables.

gs_dump **-W** *password* **-U** *jack* **-f** */home//backup/MPPDB_table_backup* **-p** 8000 *-h 10.10.10.100 human_resource* **-t** *hr.staffs* **-F** *d*

Table 4-17 Common parameters

Parameter	Description	Example Value
-U	Username for connecting to the database. If this parameter is not configured, the username of the connected database is used.	-U jack
-W	User password for database connection. ● This parameter is not required for database administrators if the trust policy is used for authentication. ● If you connect to the database without specifying this parameter and you are not a database administrator, you will be prompted to enter the password.	-W *password*
-f	Folder to store exported files. If this parameter is not specified, the exported files are stored in the standard output.	-f /home//backup/ MPPDB_table_backup
-p	Name extension of the TCP port on which the server is listening or the local Unix domain socket. This parameter is configured to ensure connections.	-p 8000

Parameter	Description	Example Value
-h	*Cluster address*. If a public network address is used for connection, set this parameter to **Public Network Address** or **Public Network Domain Name**. If a private network address is used for connection, set this parameter to **Private Network Address** or **Private Network Domain Name**.	-h 10.10.10.100
dbname	Name of the database to be exported.	human_resource
-t	Table (or view, sequence, foreign table) to be exported. You can specify multiple tables by listing them or using wildcard characters. When you use wildcard characters, quote wildcard patterns with single quotation marks (") to prevent the shell from expanding the wildcard characters. ● Single table: Enter **-t** *schema.table*. ● Multiple tables: Enter **-t** *schema.table* for each table.	● Single table: **-t hr.staffs** ● Multiple tables: **-t hr.staffs -t hr.employments**
-F	Format of exported files. The values of **-F** are as follows: ● **p**: plain text ● **c**: custom ● **d**: directory ● **t**: .tar	-F d

For details about other parameters, see "gs_dump" in the *Tool Guide*.

----End

Examples

Example 1: Use gs_dump to run the following command to export full information of the **hr.staffs** table. The exported files are in text format.
```
gs_dump -W password -U jack -f /home//backup/MPPDB_table_backup.sql -p 8000 -h 10.10.10.100
human_resource -t hr.staffs -Z 6 -F p
gs_dump[port="][human_resource][2017-07-21 17:05:10]: dump database human_resource successfully
gs_dump[port="][human_resource][2017-07-21 17:05:10]: total time: 3116  ms
```

Example 2: Use gs_dump to run the following command to export data of the **hr.staffs** table. The exported files are in .tar format.

```
gs_dump -W password -U jack -f /home//backup/MPPDB_table_data_backup.tar -p 8000 -h 10.10.10.100
human_resource -t hr.staffs -a -F t
gs_dump[port="][human_resource][2017-07-21 17:04:26]: dump database human_resource successfully
gs_dump[port="][human_resource][2017-07-21 17:04:26]: total time: 2570  ms
```

Example 3: Use gs_dump to run the following command to export the definition of the **hr.staffs** table. The exported files are stored in a directory.

```
gs_dump -W password -U jack -f /home//backup/MPPDB_table_def_backup -p 8000 -h 10.10.10.100
human_resource -t hr.staffs -s -F d
gs_dump[port="][human_resource][2017-07-21 17:03:09]: dump database human_resource successfully
gs_dump[port="][human_resource][2017-07-21 17:03:09]: total time: 2297  ms
```

Example 4: Use gs_dump to run the following command to export the **human_resource** database excluding the **hr.staffs** table. The exported files are in a custom format.

```
gs_dump -W password -U jack -f /home/backup/MPPDB_table_backup4.dmp -p 8000 -h 10.10.10.100
human_resource -T hr.staffs -F c
gs_dump[port="][human_resource][2017-07-21 17:14:11]: dump database human_resource successfully
gs_dump[port="][human_resource][2017-07-21 17:14:11]: total time: 2450  ms
```

Example 5: Use gs_dump to run the following command to export the **hr.staffs** and **hr.employments** tables. The exported files are in text format.

```
gs_dump -W password -U jack -f /home/backup/MPPDB_table_backup1.sql -p 8000 -h 10.10.10.100
human_resource -t hr.staffs -t hr.employments -F p
gs_dump[port="][human_resource][2017-07-21 17:19:42]: dump database human_resource successfully
gs_dump[port="][human_resource][2017-07-21 17:19:42]: total time: 2414  ms
```

Example 6: Use gs_dump to run the following command to export the **human_resource** database excluding the **hr.staffs** and **hr.employments** tables. The exported files are in text format.

```
gs_dump -W password -U jack -f /home/backup/MPPDB_table_backup2.sql -p 8000 -h 10.10.10.100
human_resource -T hr.staffs -T hr.employments -F p
gs_dump[port="][human_resource][2017-07-21 17:21:02]: dump database human_resource successfully
gs_dump[port="][human_resource][2017-07-21 17:21:02]: total time: 3165  ms
```

Example 7: Use gs_dump to run the following command to export data and definition of the **hr.staffs** table, and the definition of the **hr.employments** table. The exported files are in .tar format.

```
gs_dump -W password -U jack -f /home/backup/MPPDB_table_backup3.tar -p 8000 -h 10.10.10.100
human_resource -t hr.staffs -t hr.employments --exclude-table-data hr.employments -F t
gs_dump[port="][human_resource][2018-11-14 11:32:02]: dump database human_resource successfully
gs_dump[port="][human_resource][2018-11-14 11:32:02]: total time: 1645  ms
```

Example 8: Use gs_dump to run the following command to export data and definition of the **hr.staffs** table, encrypt the exported files, and store them in text format.

```
gs_dump -W password -U jack -f /home/backup/MPPDB_table_backup4.sql -p 8000 -h 10.10.10.100
human_resource -t hr.staffs --with-encryption AES128 --with-key 1212121212121212 -F p
gs_dump[port="][human_resource][2018-11-14 11:35:30]: dump database human_resource successfully
gs_dump[port="][human_resource][2018-11-14 11:35:30]: total time: 6708  ms
```

Example 9: Use gs_dump to run the following command to export all tables, including views, sequences, and foreign tables, in the **public** schema, and the **staffs** table in the **hr** schema, including data and table definition. The exported files are in a custom format.

```
gs_dump -W password -U jack -f /home/backup/MPPDB_table_backup5.dmp -p 8000 -h 10.10.10.100
human_resource -t public.* -t hr.staffs -F c
gs_dump[port="][human_resource][2018-12-13 09:40:24]: dump database human_resource successfully
gs_dump[port="][human_resource][2018-12-13 09:40:24]: total time: 896  ms
```

Example 10: Use gs_dump to run the following command to export the definition of the view referencing to the **test1** table in the **t1** schema. The exported files are in a custom format.

```
gs_dump -W password -U jack -f /home//backup/MPPDB_view_backup6 -p 8000 -h 10.10.10.100
human_resource -t t1.test1 --include-depend-objs --exclude-self -F d
gs_dump[port="][jack][2018-11-14 17:21:18]: dump database human_resource successfully
gs_dump[port="][jack][2018-11-14 17:21:23]: total time: 4239  ms
```

4.4.1.3 Exporting All Databases

4.4.1.3.1 Exporting All Databases

You can use gs_dumpall to export full information of all databases in a cluster from GaussDB(DWS), including information about each database and global objects in the cluster. You can specify the information to be exported as follows:

- Export full information of all databases, including information about each database and global objects (such as roles and tablespaces) in the cluster.

  You can use the exported information to create a same cluster containing the same databases, global objects, and data as the current one.

- Export data of all databases, excluding all object definitions and global objects.

- Export all object definitions of all databases, including the definitions of tablespaces, databases, functions, schemas, tables, indexes, and stored procedures.

  You can use the exported object definitions to quickly create a same cluster as the current one, containing the same databases and tablespaces but without data.

Procedure

Step 1 **Preparing an ECS as the gsql Client Host**.

Step 2 Download the gsql client and use an SSH transfer tool (such as WinSCP) to upload it to the Linux server where gsql is to be installed. For details, see **Downloading the Client**.

The user who uploads the client must have the full control permission on the target directory on the host to which the client is uploaded.

Alternatively, you can remotely log in to the Linux host where the gsql is to be installed in SSH mode and run the following command in the Linux command window to download the gsql client:

```
wget https://obs.myhuaweicloud.com/dws/download/dws_client_8.x.x_redhat_x64.zip --no-check-certificate
```

Step 3 Run the following commands to decompress the client:

```
cd <Path_for_storing_the_client>
unzip dws_client_8.x.x_redhat_x64.zip
```

Where,

- *<Path_for_storing_the_client>*: Replace it with the actual path.

- *dws_client_8.1.x_redhat_x86.zip*: This is the client tool package of **RedHat x86**. Replace it with the actual one.

Step 4 Run the following command to configure the GaussDB(DWS) client:

```
source gsql_env.sh
```

If the following information is displayed, the GaussDB(DWS) client is successfully configured:

```
All things done.
```

Step 5 Use gs_dumpall to run the following command to export information of all databases.

```
gs_dumpall -W password -U dbadmin -f /home/dbadmin/backup/MPPDB_backup.sql -p 8000 -h 10.10.10.100
```

Table 4-18 Common parameters

Parameter	Description	Example Value
-U	Username for database connection. The user must be a cluster administrator.	-U dbadmin
-W	User password for database connection. ● This parameter is not required for database administrators if the trust policy is used for authentication. ● If you connect to the database without specifying this parameter and you are not a database administrator, you will be prompted to enter the password.	-W *Password*
-f	Folder to store exported files. If this parameter is not specified, the exported files are stored in the standard output.	-f /home/dbadmin/backup/ MPPDB_backup.sql
-p	Name extension of the TCP port on which the server is listening or the local Unix domain socket. This parameter is configured to ensure connections.	-p 8000
-h	*Cluster address*: If a public network address is used for connection, set this parameter to **Public Network Address** or **Public Network Domain Name**. If a private network address is used for connection, set this parameter to **Private Network Address** or **Private Network Domain Name**.	-h 10.10.10.100

For details about other parameters, see "gs_dumpall" in the *Tool Guide*.

----End

Examples

Example 1: Use **gs_dumpall** to run the following command as the cluster administrator **dbadmin** to export information of all databases in a cluster. The exported files are in text format. After the command is executed, a large amount of output information will be displayed. **total time** will be displayed at the end of the information, indicating that the export is successful. In this example, only related output information is included.

```
gs_dumpall -W password -U dbadmin -f /home/dbadmin/backup/MPPDB_backup.sql -p 8000 -h
10.10.10.100
gs_dumpall[port=''][2017-07-21 15:57:31]: dumpall operation successful
gs_dumpall[port=''][2017-07-21 15:57:31]: total time: 9627  ms
```

Example 2: Use **gs_dumpall** to run the following command as the cluster administrator **dbadmin** to export definitions of all databases in a cluster. The exported files are in text format. After the command is executed, a large amount of output information will be displayed. **total time** will be displayed at the end of the information, indicating that the export is successful. In this example, only related output information is included.

```
gs_dumpall -W password -U dbadmin -f /home/dbadmin/backup/MPPDB_backup.sql -p 8000 -h
10.10.10.100 -s
gs_dumpall[port=''][2018-11-14 11:28:14]: dumpall operation successful
gs_dumpall[port=''][2018-11-14 11:28:14]: total time: 4147  ms
```

Example 3: Use gs_dumpall to run the following command export data of all databases in a cluster, encrypt the exported files, and store them in text format. After the command is executed, a large amount of output information will be displayed. **total time** will be displayed at the end of the information, indicating that the export is successful. In this example, only related output information is included.

```
gs_dumpall -W password -U dbadmin -f /home/dbadmin/backup/MPPDB_backup.sql -p 8000 -h
10.10.10.100 -a --with-encryption AES128 --with-key 1234567812345678
gs_dumpall[port=''][2018-11-14 11:32:26]: dumpall operation successful
gs_dumpall[port=''][2018-11-14 11:23:26]: total time: 4147  ms
```

4.4.1.3.2 Exporting Global Objects

You can use gs_dumpall to export global objects from GaussDB(DWS), including database users, user groups, tablespaces, and attributes (for example, global access permissions).

Procedure

Step 1 **Preparing an ECS as the gsql Client Host**.

Step 2 Download the gsql client and use an SSH transfer tool (such as WinSCP) to upload it to the Linux server where gsql is to be installed. For details, see **Downloading the Client**.

The user who uploads the client must have the full control permission on the target directory on the host to which the client is uploaded.

Alternatively, you can remotely log in to the Linux host where the gsql is to be installed in SSH mode and run the following command in the Linux command window to download the gsql client:

```
wget https://obs.myhuaweicloud.com/dws/download/dws_client_8.x.x_redhat_x64.zip --no-check-certificate
```

Step 3 Run the following commands to decompress the client:

```
cd <Path_for_storing_the_client>
unzip dws_client_8.x.x_redhat_x64.zip
```

Where,

- *<Path_for_storing_the_client>*: Replace it with the actual path.

- *dws_client_8.1.x_redhat_x86.zip*: This is the client tool package of **RedHat x86**. Replace it with the actual one.

Step 4 Run the following command to configure the GaussDB(DWS) client:

```
source gsql_env.sh
```

If the following information is displayed, the GaussDB(DWS) client is successfully configured:

```
All things done.
```

Step 5 Use gs_dumpall to run the following command to export tablespace objects.

```
gs_dumpall -W password -U dbadmin -f /home/dbadmin/backup/MPPDB_tablespace.sql -p 8000 -h
10.10.10.100 -t
```

Table 4-19 Common parameters

Parameter	Description	Example Value
-U	Username for database connection. The user must be a cluster administrator.	-U dbadmin
-W	User password for database connection. ● This parameter is not required for database administrators if the trust policy is used for authentication. ● If you connect to the database without specifying this parameter and you are not a database administrator, you will be prompted to enter the password.	-W *Password*
-f	Folder to store exported files. If this parameter is not specified, the exported files are stored in the standard output.	-f /home//backup/ *MPPDB_tablespace*.sql

Parameter	Description	Example Value
-p	Name extension of the TCP port on which the server is listening or the local Unix domain socket. This parameter is configured to ensure connections.	-p 8000
-h	*Cluster address*: If a public network address is used for connection, set this parameter to **Public Network Address** or **Public Network Domain Name**. If a private network address is used for connection, set this parameter to **Private Network Address** or **Private Network Domain Name**.	-h 10.10.10.100
-t	Dumps only tablespaces. You can also use **--tablespaces-only** alternatively.	-

For details about other parameters, see "gs_dumpall" in the *Tool Guide*.

----End

Examples

Example 1: Use **gs_dumpall** to run the following command as the cluster administrator **dbadmin** to export information of global tablespaces and users in a cluster. The exported files are in text format.

```
gs_dumpall -W password -U dbadmin -f /home/dbadmin/backup/MPPDB_globals.sql -p 8000 -h
10.10.10.100 -g
gs_dumpall[port=''][2018-11-14 19:06:24]: dumpall operation successful
gs_dumpall[port=''][2018-11-14 19:06:24]: total time: 1150  ms
```

Example 2: Use **gs_dumpall** to run the following command as the cluster administrator **dbadmin** to export global tablespaces in a cluster, encrypt the exported files, and store them in text format.

```
gs_dumpall -W password -U dbadmin -f /home/dbadmin/backup/MPPDB_tablespace.sql -p 8000 -h
10.10.10.100 -t --with-encryption AES128 --with-key 1212121212121212
gs_dumpall[port=''][2018-11-14 19:00:58]: dumpall operation successful
gs_dumpall[port=''][2018-11-14 19:00:58]: total time: 186  ms
```

Example 3: Use **gs_dumpall** to run the following command as the cluster administrator **dbadmin** to export information of global users in a cluster. The exported files are in text format.

```
gs_dumpall -W password -U dbadmin -f /home/dbadmin/backup/MPPDB_user.sql -p 8000 -h 10.10.10.100 -
r
gs_dumpall[port=''][2018-11-14 19:03:18]: dumpall operation successful
gs_dumpall[port=''][2018-11-14 19:03:18]: total time: 162  ms
```

4.4.1.4 Data Export By a User Without Required Permissions

gs_dump and gs_dumpall use **-U** to specify the user that performs the export. If the specified user does not have the required permission, data cannot be exported. In this case, you can set **--role** in the export command to the role that has the permission. Then, gs_dump or gs_dumpall uses the specified role to export data.

Procedure

Step 1 **Preparing an ECS as the gsql Client Host**.

Step 2 Download the gsql client and use an SSH transfer tool (such as WinSCP) to upload it to the Linux server where gsql is to be installed. For details, see **Downloading the Client**.

The user who uploads the client must have the full control permission on the target directory on the host to which the client is uploaded.

Alternatively, you can remotely log in to the Linux host where the gsql is to be installed in SSH mode and run the following command in the Linux command window to download the gsql client:

wget https://obs.myhuaweicloud.com/dws/download/dws_client_8.x.x_redhat_x64.zip --no-check-certificate

Step 3 Run the following commands to decompress the client:

cd <*Path_for_storing_the_client*>
unzip *dws_client_8.x.x_redhat_x64.zip*

Where,

● <*Path_for_storing_the_client*>: Replace it with the actual path.

● *dws_client_8.1.x_redhat_x86.zip*: This is the client tool package of **RedHat x86**. Replace it with the actual one.

Step 4 Run the following command to configure the GaussDB(DWS) client:

source gsql_env.sh

If the following information is displayed, the GaussDB(DWS) client is successfully configured:

All things done.

Step 5 Use gs_dump to export data of the **human_resource** database.

User **jack** does not have the permission for exporting data of the **human_resource** database and the role **role1** has this permission. To export data of the **human_resource** database, you can set **--role** to **role1** in the export command. The exported files are in .tar format.

gs_dump -U *jack* **-W** *password* **-f** */home//backup/MPPDB_backup.tar* **-p** 8000 *-h 10.10.10.100 human_resource* **--role** *role1* **--rolepassword** *password* **-F** *t*

Table 4-20 Common parameters

Parameter	Description	Example Value (dbadmin)
-U	Username for database connection.	-U jack

Parameter	Description	Example Value (dbadmin)
-W	User password for database connection. ● This parameter is not required for database administrators if the trust policy is used for authentication. ● If you connect to the database without specifying this parameter and you are not a database administrator, you will be prompted to enter the password.	-W *Password*
-f	Folder to store exported files. If this parameter is not specified, the exported files are stored in the standard output.	-f /home//backup/ MPPDB_backup.tar
-p	Name extension of the TCP port on which the server is listening or the local Unix domain socket. This parameter is configured to ensure connections.	-p 8000
-h	*Cluster address*: If a public network address is used for connection, set this parameter to **Public Network Address** or **Public Network Domain Name**. If a private network address is used for connection, set this parameter to **Private Network Address** or **Private Network Domain Name**.	-h 10.10.10.100
dbname	Name of the database to be exported.	human_resource
--role	Role name for the export operation. After this parameter is set and gs_dump or gs_dumpall connects to the database, the **SET ROLE** command will be issued. When the user specified by **-U** does not have the permissions required by gs_dump or gs_dumpall, this parameter allows the user to switch to a role with the required permissions.	-r role1

Parameter	Description	Example Value (dbadmin)
--rolepassword	Role password.	--rolepassword password
-F	Format of exported files. The values of **-F** are as follows: ● **p**: plain text ● **c**: custom ● **d**: directory ● **t**: .tar	-F t

For details about other parameters, see "gs_dump" or "gs_dumpall" in the *Tool Guide*.

----End

Examples

Example 1: User **jack** does not have the permission for exporting data of the **human_resource** database and the role **role1** has this permission. To export data of the **human_resource** database, you can set **--role** to **role1** in the export command. The exported files are in .tar format.

```
human_resource=# CREATE USER jack IDENTIFIED BY "password";

gs_dump -U jack -W password -f /home//backup/MPPDB_backup11.tar -p 8000 -h 10.10.10.100
human_resource --role role1 --rolepassword password -F t
gs_dump[port='8000'][human_resource][2017-07-21 16:21:10]: dump database human_resource successfully
gs_dump[port='8000'][human_resource][2017-07-21 16:21:10]: total time: 4239  ms
```

Example 2: User **jack** does not have the permission for exporting the **public** schema and the role **role1** has this permission. To export the **public** schema, you can set **--role** to **role1** in the export command. The exported files are in .tar format.

```
human_resource=# CREATE USER jack IDENTIFIED BY "1234@abc";

gs_dump -U jack -W password -f /home//backup/MPPDB_backup12.tar -p 8000 -h 10.10.10.100
human_resource -n public --role role1 --rolepassword password -F t
gs_dump[port='8000'][human_resource][2017-07-21 16:21:10]: dump database human_resource successfully
gs_dump[port='8000'][human_resource][2017-07-21 16:21:10]: total time: 3278  ms
```

Example 3: User **jack** does not have the permission for exporting all databases in a cluster and the role **role1** has this permission. To export all databases, you can set **--role** to **role1** in the export command. The exported files are in text format.

```
human_resource=# CREATE USER jack IDENTIFIED BY "password";

gs_dumpall -U jack -W password -f /home//backup/MPPDB_backup.sql -p 8000 -h 10.10.10.100 --role role1
--rolepassword password
gs_dumpall[port='8000'][human_resource][2018-11-14 17:26:18]: dumpall operation successful
gs_dumpall[port='8000'][human_resource][2018-11-14 17:26:18]: total time: 6437  ms
```

4.4.2 Using gs_restore to Import Data

Scenarios

gs_restore is an import tool provided by GaussDB(DWS). You can use **gs_restore** to import the files exported by **gs_dump** to a database. **gs_restore** can import the files in .tar, custom, or directory format.

gs_restore can:

- Import data to a database.

  If a database is specified, data is imported to the database. If multiple databases are specified, the password for connecting to each database also needs to be specified.

- Import data to a script.

  If no database is specified, a script containing the SQL statement to recreate the database is created and written to a file or standard output. This script output is equivalent to the plain text output of **gs_dump**.

You can specify and sort the data to be imported.

Procedure

📖 NOTE

gs_restore incrementally imports data by default. To prevent data exception caused by consecutive imports, use the **-e** and **-c** parameters for each import. In this way, existing data is deleted from the target database before each import; the system exists the import task with an error (error message is displayed after the import process is complete) and proceeds with the next.

Step 1 Log in to the server as the **root** user and run the following command to go to the data storage path:

```
cd /opt/bin
```

Step 2 Use **gs_restore** to import all object definitions from the exported file of the whole **postgres** database to the **backupdb** database.

```
gs_restore -W password -U jack /home//backup/MPPDB_backup.tar -p 8000 -h 10.10.10.100 -d backupdb -s -e -c
```

Table 4-21 Common parameters

Parameter	Description	Example Value
-U	Username for database connection.	-U jack

Parameter	Description	Example Value
-W	User password for database connection. ● This parameter is not required for database administrators if the trust policy is used for authentication. ● If you connect to the database without specifying this parameter and you are not a database administrator, you will be prompted to enter the password.	-W *Password*
-d	Database to which data will be imported.	-d backupdb
-p	TCP port or the local Unix-domain socket file extension on which the server is listening for connections.	-p 8000
-h	*Cluster address*: If a public network address is used for connection, set this parameter to **Public Network Address** or **Public Network Domain Name**. If a private network address is used for connection, set this parameter to **Private Network Address** or **Private Network Domain Name**.	-h 10.10.10.100
-e	Exits the current import task and performs the next if an error occurs when you send a SQL statement in the current import task. Error messages are displayed after the import process is complete.	-
-c	Cleans existing objects from the target database before the import.	-
-s	Imports only object definitions in schemas and does not import data. Sequence values will also not be imported.	-

For details about other parameters, see "Server Tools > gs_restore" in the *Tool Reference*.

----End

Examples

Example 1: Run **gs_restore** to import data and all object definitions of the **postgres** database from the **MPPDB_backup.dmp** file (custom format).

```
gs_restore -W password backup/MPPDB_backup.dmp -p 8000 -h 10.10.10.100 -d backupdb
gs_restore[2017-07-21 19:16:26]: restore operation successfu
gs_restore: total time: 13053  ms
```

Example 2: Run **gs_restore** to import data and all object definitions of the **postgres** database from the **MPPDB_backup.tar** file.

```
gs_restore backup/MPPDB_backup.tar -p 8000 -h 10.10.10.100 -d backupdb
gs_restore[2017-07-21 19:21:32]: restore operation successful
gs_restore[2017-07-21 19:21:32]: total time: 21203  ms
```

Example 3: Run **gs_restore** to import data and all object definitions of the **postgres** database from the **MPPDB_backup** directory.

```
gs_restore backup/MPPDB_backup -p 8000 -h 10.10.10.100 -d backupdb
gs_restore[2017-07-21 19:26:46]: restore operation successful
gs_restore[2017-07-21 19:26:46]: total time: 21003  ms
```

Example 4: Run **gs_restore** to import all object definitions of the **postgres** database from the **MPPDB_backup.tar** file. Table data is not imported.

```
gs_restore -W password /home//backup/MPPDB_backup.tar -p 8000 -h 10.10.10.100 -d backupdb -s -e -c
gs_restore[2017-07-21 19:46:27]: restore operation successful
gs_restore[2017-07-21 19:46:27]: total time: 32993  ms
```

Example 5: Run **gs_restore** to import data and all definitions in the **PUBLIC** schema from the **MPPDB_backup.dmp** file. Existing objects are deleted from the target database before the import. If an existing object references to an object in another schema, you need to manually delete the referenced object first.

```
gs_restore backup/MPPDB_backup.dmp -p 8000 -h 10.10.10.100 -d backupdb -e -c -n PUBLIC
gs_restore: [archiver (db)] Error while PROCESSING TOC:
gs_restore: [archiver (db)] Error from TOC entry 313; 1259 337399 TABLE table1 gaussdba
gs_restore: [archiver (db)] could not execute query: ERROR:  cannot drop table table1 because other objects
depend on it
DETAIL:  view t1.v1 depends on table table1
HINT:  Use DROP ... CASCADE to drop the dependent objects too.
Command was: DROP TABLE public.table1;
```

Manually delete the referenced object and create it again after the import is complete.

```
gs_restore backup/MPPDB_backup.dmp -p 8000 -h 10.10.10.100 -d backupdb -e -c -n PUBLIC
gs_restore[2017-07-21 19:52:26]: restore operation successful
gs_restore[2017-07-21 19:52:26]: total time: 2203  ms
```

Example 6: Run **gs_restore** to import the definition of the **hr.staffs** table in the **PUBLIC** schema from the **MPPDB_backup.dmp** file. Before the import, the **hr.staffs** table does not exist.

```
gs_restore backup/MPPDB_backup.dmp -p 8000 -h 10.10.10.100 -d backupdb -e -c -s -n PUBLIC -t hr.staffs
gs_restore[2017-07-21 19:56:29]: restore operation successful
gs_restore[2017-07-21 19:56:29]: total time: 21000  ms
```

Example 7: Run **gs_restore** to import data of the **hr.staffs** table in **PUBLIC** schema from the **MPPDB_backup.dmp** file. Before the import, the **hr.staffs** table is empty.

```
gs_restore backup/MPPDB_backup.dmp -p 8000 -h 10.10.10.100 -d backupdb -e -a -n PUBLIC -t hr.staffs
gs_restore[2017-07-21 20:12:32]: restore operation successful
gs_restore[2017-07-21 20:12:32]: total time: 20203  ms
```

Example 8: Run **gs_restore** to import the definition of the **hr.staffs** table. Before the import, the **hr.staffs** table already exists.

```
human_resource=# select * from hr.staffs;
 staff_id | first_name | last_name | email   | phone_number   |   hire_date      | employment_id |
salary  | commission_pct | manager_id | section_id
----------+------------+------------+----------+--------------------+--------------------+---------------+----------
+----------------+------------+------------
     200 | Jennifer   | Whalen     | JWHALEN  | 515.123.4444      | 1987-09-17 00:00:00 | AD_ASST       |
4400.00 |                |        101 |       10
     201 | Michael    | Hartstein  | MHARTSTE | 515.123.5555      | 1996-02-17 00:00:00 | MK_MAN        |
13000.00 |                |        100 |       20

gsql -d human_resource -p 8000
gsql ((GaussDB 8.1.3 build 39137c2d) compiled at 2022-04-01 15:43:11 commit 3629 last mr 5138 release)
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

human_resource=# drop table hr.staffs CASCADE;
NOTICE:  drop cascades to view hr.staff_details_view

gs_restore -W password /home//backup/MPPDB_backup.tar -p 8000 -h 10.10.10.100-d human_resource -n
hr -t staffs -s -e
restore operation successful
total time: 904  ms

human_resource=# select * from hr.staffs;
 staff_id | first_name | last_name | email  | phone_number | hire_date | employment_id | salary |
commission_pct | manager_id | section_id
----------+------------+-----------+-------+--------------+-----------+---------------+--------+----------------
+------------+------------
(0 rows)
```

Example 9: Run **gs_restore** to import data and definitions of the **staffs** and **areas** tables. Before the import, the **staffs** and **areas** tables do not exist.

```
human_resource=# \d
                 List of relations
 Schema |       Name        | Type | Owner |           Storage
--------+-------------------+------+-------+--------------------------------
 hr     | employment_history | table | | {orientation=row,compression=no}
 hr     | employments       | table | | {orientation=row,compression=no}
 hr     | places            | table | | {orientation=row,compression=no}
 hr     | sections          | table | | {orientation=row,compression=no}
 hr     | states            | table | | {orientation=row,compression=no}
(5 rows)

gs_restore -W password /home/mppdb/backup/MPPDB_backup.tar -p 8000 -h 10.10.10.100 -d
human_resource -n hr -t staffs -n hr -t areas
restore operation successful
total time: 724  ms

human_resource=# \d
                 List of relations
 Schema |       Name        | Type | Owner |           Storage
--------+-------------------+------+-------+--------------------------------
 hr     | areas             | table | | {orientation=row,compression=no}
 hr     | employment_history | table | | {orientation=row,compression=no}
 hr     | employments       | table | | {orientation=row,compression=no}
 hr     | places            | table | | {orientation=row,compression=no}
 hr     | sections          | table | | {orientation=row,compression=no}
 hr     | staffs            | table | | {orientation=row,compression=no}
 hr     | states            | table | | {orientation=row,compression=no}
(7 rows)
```

```
human_resource=# select * from hr.areas;
 area_id |     area_name
---------+------------------------
      4 | Iron
      1 | Wood
      2 | Lake
      3 | Desert
(4 rows)
```

Example 10: Run **gs_restore** to import data and all object definitions in the **hr** schema.

```
gs_restore -W password /home//backup/MPPDB_backup1.sql -p 8000 -h 10.10.10.100 -d backupdb -n hr -e -
c
restore operation successful
total time: 702  ms
```

Example 11: Run **gs_restore** to import all object definitions in the **hr** and **hr1** schemas to the **backupdb** database.

```
gs_restore -W password /home//backup/MPPDB_backup2.dmp -p 8000 -h 10.10.10.100 -d backupdb -n hr -
n hr1 -s
restore operation successful
total time: 665  ms
```

Example 12: Run **gs_restore** to decrypt the files exported from the **human_resource** database and import them to the **backupdb** database.

```
create database backupdb;


gs_restore /home//backup/MPPDB_backup.tar -p 8000 -h 10.10.10.100 -d backupdb --with-
key=1234567812345678
restore operation successful
total time: 23472  ms

gsql -d backupdb -p 8000 -r
gsql ((GaussDB 8.1.3 build 39137c2d) compiled at 2022-04-01 15:43:11 commit 3629 last mr 5138 release)
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

backupdb=# select * from hr.areas;
 area_id |     area_name
---------+------------------------
      4 | Iron
      1 | Wood
      2 | Lake
      3 | Desert
(4 rows)
```

Example 13: **user 1** does not have the permission to import data from an exported file to the **backupdb** database and **role1** has this permission. To import the exported data to the **backupdb** database, you can set **--role** to **role1** in the **gs_restore** command.

```
human_resource=# CREATE USER user1 IDENTIFIED BY 'password';

gs_restore -U user1 -W password /home//backup/MPPDB_backup.tar -p 8000 -h 10.10.10.100 -d backupdb
--role role1 --rolepassword password
restore operation successful
total time: 554  ms

gsql -d backupdb -p 8000 -r
gsql ((GaussDB 8.1.3 build 39137c2d) compiled at 2022-04-01 15:43:11 commit 3629 last mr 5138 release)
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

backupdb=# select * from hr.areas;
 area_id |     area_name
```

```
---------+-----------------------
    4 | Iron
    1 | Wood
    2 | Lake
    3 | Desert
(4 rows)
```

4.5 Exporting Data

4.5.1 Exporting Data to OBS

4.5.1.1 Parallel OBS Data Export

Overview

GaussDB(DWS) databases allow you to export data in parallel using OBS foreign tables, in which the export mode and the exported data format are specified. Data is exported in parallel through multiple DNs from GaussDB(DWS) to the OBS server, improving the overall export performance.

- The CN only plans data export tasks and delivers the tasks to DNs. In this case, the CN is released to process external requests.

- The computing capability and bandwidth of all the DNs are fully leveraged to export data.

- You can concurrently export data using multiple OBS services, but the bucket and object paths specified for the export tasks must be different and cannot be null.

- The OBS server connects to GaussDB(DWS) cluster nodes. The export rate is affected by the network bandwidth.

- The TEXT and CSV data file formats are supported. The size of data in a single row must be less than 1 GB.

- Data in ORC format is supported only by 8.1.0 or later.

Related Concepts

- **Source data file**: a TEXT or CSV file that stores data.

- **OBS**: a cloud storage service used to store unstructured data, such as documents, images, and videos. Data objects concurrently exported from GaussDB(DWS) are stored on the OBS server.

- **Bucket**: a container storing objects on OBS.

  - Object storage is a flat storage mode. Layered file system structures are not needed because all objects in buckets are at the same logical layer.

  - In OBS, each bucket name must be unique and cannot be changed. A default access control list (ACL) is created with a bucket. Each item in the ACL contains permissions granted to certain users, such as **READ**, **WRITE**, and **FULL_CONTROL**. Only authorized users can perform bucket operations, such as creating, deleting, viewing, and setting ACLs for buckets.

  - A user can create a maximum of 100 buckets. The total data size and the number of objects and files in each bucket are not limited.

- **Object**: a basic data storage unit in OBS. Data uploaded by users is stored in OBS buckets as objects. Object attributes include **Key**, **Metadata**, and **Data**.

  Generally, objects are managed as files. However, OBS has no file system–related concepts, such as files or folders. To let users easily manage data, OBS allows them to simulate folders. Users can add a slash (/) in the object name, for example, **tpcds1000/stock.csv**. In this name, **tpcds1000** is regarded as the folder name and **stock.csv** the file name. The value of **key** (object name) is still **tpcds1000/stock.csv**, and the content of the object is the content of the **stock.csv** file.

- **Key**: name of an object. It is a UTF-8 character sequence containing 1 to 1024 characters. A key value must be unique in a bucket. Users can name the objects they stored or obtained as *Bucket name*+*Object name*.

- **Metadata**: object metadata, which contains information about the object. There are system metadata and user metadata. The metadata is uploaded to OBS as key-value pairs together with HTTP headers.

  - System metadata is generated by OBS and used for processing object data. System metadata includes **Date**, **Content-length**, **last-modify**, and **Content-MD5**.

  - User metadata contains object descriptions specified by users for uploading objects.

- **Data**: object content, which is regarded by OBS as stateless binary data.

- **Foreign table**: A foreign table is used to identify data in a source data file. It stores information, such as the location, format, destination location, encoding format, and data delimiter of a source data file.

Principles

The following describes the principles of exporting data from a cluster to OBS by using a distributed hash table or a replication table.

- Distributed hash table: the table for which **DISTRIBUTE BY HASH (Column_Name)** is specified in the table creation statement.

  A distributed hash table stores data in hash mode. **Figure 4-9** shows how to export data from table (**T2**) to OBS as an example.

  During table data storage, the **col2** hash column in table **T2** is hashed, and a hash value is generated. The tuple is distributed to corresponding DNs for storage according to the mapping between the DNs and the hash value.

  When data is exported to OBS, DNs that store the exported data of **T2** directly export their data files to OBS. Original data on multiple nodes will be exported in parallel.

Figure 4-9 Hash distribution principle
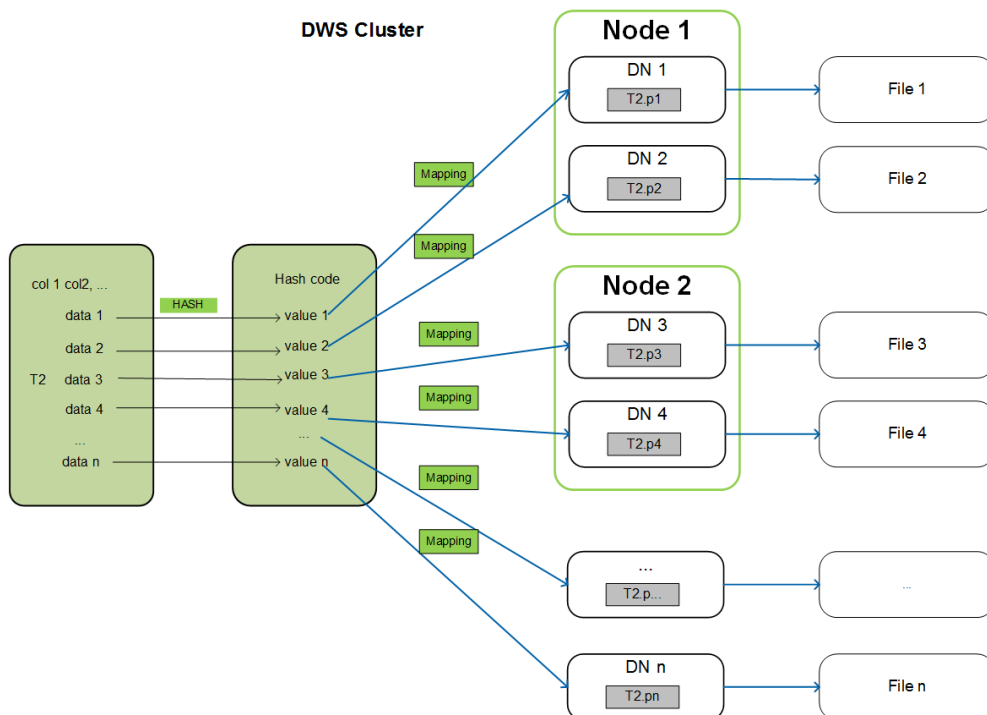
- Replication table: the table for which **DISTRIBUTE BY REPLICATION** is specified in the table creation statement.

  A replication table stores a package of complete table data on each GaussDB(DWS) node. When exporting data to OBS, GaussDB(DWS) randomly selects a DN for export.

Naming Rules of Exported Files

Rules for naming the files exported from GaussDB(DWS) to OBS are as follows:

- Data exported from DNs is stored on OBS in segment format. The file is named as *Table name_Node name_***segment.***n*. *n* is a natural number starting from 0, for example, 0, 1, 2, 3.

  For example, the data of table **t1** on **datanode3** will be exported as **t1_datanode3_segment.0**, **t1_datanode3_segment.1**, and so on.

  You are advised to export data from different clusters or databases to different OBS buckets or different paths of the same OBS bucket.

- Each segment can store a maximum of 1 GB data, with no tuples sliced. If data stored in a segment exceeds 1 GB, the excess data will be stored in the second segment.

  For example:

  A segment has already stored 100 pieces of tuples (1023 MB) when **datanode3** exports data from **t1** to OBS. If a 5 MB tuple is inserted to the segment, the data size becomes 1028 MB. In this case, file **t1_datanode3_segment.0** (1023 MB) is generated and stored on OBS, and the new tuple is stored on OBS as file **t1_datanode3_segment.1**.

- When data is exported from a distributed hash table, the number of segments generated on each DN depends on the data volume stored on a DN, not on

the number of DNs in the cluster. Data stored in hash mode may not be evenly distributed on each DN.

For example, a cluster has **DataNode1**, **DataNode2**, **DataNode3**, **DataNode4**, **DataNode5**, and **DataNode6**, which store 1.5 GB, 0.7 GB, 0.6 GB, 0.8 GB, 0.4 GB, and 0.5 GB data, respectively. Seven OBS segment files will be generated during data export because **DataNode1** will generate two segment files, which store 1 GB and 0.5 GB data, respectively.

Data Export Process

Figure 4-10 Concurrent data export
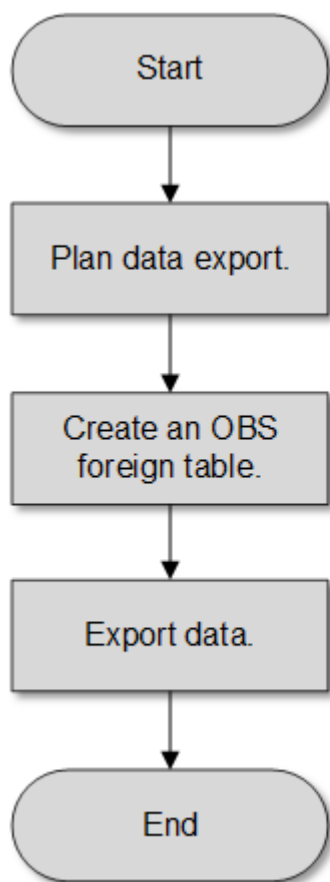
Table 4-22 Process description

Procedure	Description	Subtask
Plan data export.	Create an OBS bucket and a folder in the OBS bucket as the directory for storing exported data files. For details, see **Planning Data Export**.	-

Procedure	Description	Subtask
Create an OBS foreign table.	Create a foreign table to help OBS specify information about data files to be exported. The foreign table stores information, such as the destination location, format, encoding, and data delimiter of a source data file. For details, see **Creating an OBS Foreign Table**.	-
Export data.	After the foreign table is created, run the **INSERT** statement to efficiently export data to data files. For details, see **Exporting Data**.	-

4.5.1.2 Exporting CSV/TXT Data to OBS

4.5.1.2.1 Planning Data Export

Scenarios

Plan the storage location of exported data in OBS.

Planning OBS Save Path and File

You need to specify the OBS path (to directory) for storing data that you want to export. The exported data can be saved to a file in CSV format. The system also supports TEXT so that you can import the exported data to various applications.

The target directory cannot contain any files.

Planning OBS Bucket Permissions

The user used to export data must:

- Have OBS enabled.
- Have the write permission on the OBS bucket where the data export path is located.

  You can configure ACL permissions for the OBS bucket to grant the write permission to a specific user.

  For details, see **Granting Write Permission to OBS Storage Location and OBS Bucket as Planned**.

Planning Data to Be Exported and Foreign Tables

You must prepare data to be exported in the database table, and the data volume per row must be less than 1 GB. Based on the data to be exported, plan foreign tables whose attributes such as columns, column types, and length match those of user data.

Granting Write Permission to OBS Storage Location and OBS Bucket as Planned

Step 1 Create an OBS bucket and a folder in the OBS bucket as the directory for storing exported data.

1. Log in to the OBS management console.

   Click **Service List** and choose **Object Storage Service** to open the OBS management console.

2. Create a bucket.

   For details about how to create an OBS bucket, see **Creating a Bucket** in the *Object Storage Service Console Operation Guide*.

   For example, create two buckets named **mybucket** and **mybucket02**.

3. Create a folder.

   For details, see **Creating a Folder** in *Object Storage Service Console Operation Guide*.

   Example:

   – Create a folder named **output_data** in the **mybucket** OBS bucket.

   – Create a folder named **output_data** in the **mybucket02** OBS bucket.

Step 2 Determine the path of the created OBS folder.

Specify the OBS path for storing exported data files. This path is the value of the **location** parameter used for creating a foreign table.

The OBS folder path in the **location** parameter consists of **obs://**, a bucket name, and a file path. Example:

In this example, the OBS folder path is as follows:

```
obs://mybucket/output_data/
```

📖 **NOTE**

The OBS directory to be used for storing data files must be empty.

Step 3 Grant the OBS bucket write permission to the user who wants to export data.

When exporting data, a user must have the write permission on the OBS bucket where the data export path is located. You can configure ACL permissions for the OBS bucket to grant the write permission to a specific user.

For details, see **Configuring a Bucket ACL** in the *Object Storage Service Console Operation Guide*.

----End

4.5.1.2.2 Creating an OBS Foreign Table

Procedure

Step 1 Based on the path planned in **Planning Data Export**, determine the value of the **location** parameter used for creating a foreign table.

Step 2 Obtain the access keys (AK and SK) to access OBS.

To obtain access keys, log in to the management console, click the username in the upper right corner, and select **My Credential** from the menu. Then choose **Access Keys** in the navigation tree on the left. On the **Access Keys** page, you can view the existing AKs or click **Add Access Key** to create and download access keys.

Step 3 Examine the formats of data to be exported and determine the values of data format parameters used for creating a foreign table. For details, see data format parameters.

Step 4 Create an OBS table based on the parameter settings in the preceding steps. For details about how to create a foreign table, see CREATE FOREIGN TABLE (for GDS Import and Export).

----End

Example 1

For example, in the GaussDB(DWS) database, create a write-only foreign table with the **format** parameter as **text** to export text files. Set parameters as follows:

- **location**

  The OBS path of the source data file has been obtained in **step 2** in **Planning Data Export**.

  For example, set **location** as follows:

  ```
  location 'obs://mybucket/output_data/',
  ```

- **Access keys (AK and SK)**

  – Set **access_key** to the AK you have obtained.

  – Set **secret_access_key** to the SK you have obtained.

  📖 **NOTE**

  access_key and **secret_access_key** have been obtained during user creation. Replace the italic part with the actual keys.

- **Data format parameters**

  – Set **format** to **TEXT**.

  – Set **encoding** to **UTF-8**.

  – Configure **encrypt**. Its default value is **off**.

  – Set **delimiter** to **|**.

Based on the preceding settings, the foreign table is created using the following statements:

```
DROP FOREIGN TABLE IF EXISTS product_info_output_ext1;
CREATE FOREIGN TABLE product_info_output_ext1
(
```

```
c_bigint bigint,
c_char char(30),
c_varchar varchar(30),
c_nvarchar2 nvarchar2(30) ,
c_data date,
c_time time ,
c_test varchar(30))
server gsmpp_server
options (
LOCATION 'obs://mybucket/output_data/',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced'
format 'text',
delimiter '|',
encoding 'utf-8',
encrypt 'on'
 )
WRITE ONLY;
```

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

```
CREATE FOREIGN TABLE
```

Example 2:

For example, in the GaussDB(DWS) database, create a write-only foreign table with the **format** parameter as **CSV** to export CSV files. Set parameters as follows:

- **location**

  The OBS path of the source data file has been obtained in **step 2** in **Planning Data Export**.

  For example, set **location** as follows:

  ```
  location 'obs://mybucket/output_data/',
  ```

- **Access keys (AK and SK)**

  - Set **access_key** to the AK you have obtained.

  - Set **secret_access_key** to the SK you have obtained.

  ☐ NOTE

     access_key and **secret_access_key** have been obtained during user creation. Replace the italic part with the actual keys.

- **Data format parameters**

  - Set **format** to **CSV**.

  - Set **encoding** to **UTF-8**.

  - Configure **encrypt**. Its default value is **off**.

  - Set **delimiter** to **,**.

  - Set **header** (whether the exported data file contains the header row).

    Specifies whether a file contains a header with the names of each column in the file.

    When exporting data from OBS, this parameter cannot be set to **true**. Use the default value **false**, indicating that the first row of the exported data file is not the header.

Based on the preceding settings, the foreign table is created using the following statements:

```
DROP FOREIGN TABLE IF EXISTS product_info_output_ext2;
CREATE FOREIGN TABLE product_info_output_ext2
(
    product_price              integer        not null,
    product_id                 char(30)       not null,
    product_time               date           ,
    product_level              char(10)       ,
    product_name               varchar(200)   ,
    product_type1              varchar(20)    ,
    product_type2              char(10)       ,
    product_monthly_sales_cnt   integer        ,
    product_comment_time        date           ,
    product_comment_num         integer        ,
    product_comment_content     varchar(200)
)
SERVER gsmpp_server
OPTIONS(
location 'obs://mybucket/output_data/',
FORMAT 'CSV' ,
DELIMITER ',',
encoding 'utf8',
header 'false',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced'
)
WRITE ONLY ;
```

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

```
CREATE FOREIGN TABLE
```

4.5.1.2.3 Exporting Data

Procedure

Step 1 Export data.

```
INSERT INTO [Foreign table name] SELECT * FROM [Source table name];
```

----End

Examples

- **Example 1**: Export data from table **product_info_output** to a data file through the **product_info_output_ext** foreign table.
  ```
  INSERT INTO product_info_output_ext SELECT * FROM product_info_output;
  ```

  If information similar to the following is displayed, the data has been exported.
  ```
  INSERT 0 10
  ```

- **Example 2**: Export part of the data to a data file by specifying the filter condition **WHERE** *product_price>500*.
  ```
  INSERT INTO product_info_output_ext SELECT * FROM product_info_output WHERE product_price>500;
  ```

  ☐ NOTE

  - The directory to be used for data storage must be empty, or the export will fail.

  - Data of a special type, such as RAW, is exported as a binary file, which cannot be recognized by the import tool. You need to use the RAWTOHEX() function to convert it to the hexadecimal format before export.

4.5.1.2.4 Examples

Exporting a Table

Create two foreign tables and use them to export tables from a database to two buckets in OBS.

Step 1 Log in to the OBS data server through the management console. On the OBS server, create the buckets **/input-data1** and **/input-data2** for storing data files, and create data directories **/input-data1/data** and **/input-data2/data**, respectively, in the two buckets.

Step 2 On the GaussDB(DWS) database, create the foreign tables **tpcds.customer_address_ext1** and **tpcds.customer_address_ext2** for the OBS data server to receive data exported from the database.

OBS and the database are in the same region. The example GaussDB(DWS) table to be exported is **tpcds.customer_address**.

Export information is set as follows:

- The source data file directories are **/input-data1/data/** and **/input-data2/data/**, so **location** of **tpcds.customer_address_ext1** and **tpcds.customer_address_ext2** is set to **obs://input-data1/data/** and **obs://input-data2/data/**, respectively.

Information about data formats is set based on the detailed data format parameters specified during data export from a database. The parameter settings are as follows:

- **format** is set to **CSV**.

- **encoding** is set to **UTF-8**.

- **delimiter** is set to **E'\x08'**.

- Configure **encrypt**. Its default value is **off**.

- **access_key** is set to the AK you have obtained. (mandatory)

- **secret_access_key** is set to the SK you have obtained. (mandatory)

  📖 **NOTE**

  access_key and **secret_access_key** have been obtained during user creation. Replace the italic part with the actual keys.

Based on the preceding settings, the foreign table is created using the following statements:

```
CREATE FOREIGN TABLE tpcds.customer_address_ext1
(
ca_address_sk            integer              ,
ca_address_id            char(16)             ,
ca_street_number         char(10)             ,
ca_street_name           varchar(60)          ,
ca_street_type           char(15)             ,
ca_suite_number          char(10)             ,
ca_city                  varchar(60)          ,
ca_county                varchar(30)          ,
ca_state                 char(2)              ,
ca_zip                   char(10)             ,
ca_country               varchar(20)          ,
ca_gmt_offset            decimal(5,2)         ,
```

```
ca_location_type        char(20)
)
SERVER gsmpp_server
OPTIONS(LOCATION 'obs://input-data1/data/',
FORMAT 'CSV',
ENCODING 'utf8',
DELIMITER E'\x08',
ENCRYPT 'off',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced'
)Write Only;
CREATE FOREIGN TABLE tpcds.customer_address_ext2
(
ca_address_sk          integer              ,
ca_address_id          char(16)             ,
ca_street_number       char(10)             ,
ca_street_name         varchar(60)          ,
ca_street_type         char(15)             ,
ca_suite_number        char(10)             ,
ca_city                varchar(60)          ,
ca_county              varchar(30)          ,
ca_state               char(2)              ,
ca_zip                 char(10)             ,
ca_country             varchar(20)          ,
ca_gmt_offset          decimal(5,2)         ,
ca_location_type       char(20)
)
SERVER gsmpp_server
OPTIONS(LOCATION 'obs://input-data2/data/',
FORMAT 'CSV',
ENCODING 'utf8',
DELIMITER E'\x08',
ENCRYPT 'off',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced'
)Write Only;
```

Step 3 In GaussDB(DWS), export the data table **tpcds.customer_address** to the foreign tables **tpcds.customer_address_ext1** and **tpcds.customer_address_ext2** concurrently.

```
INSERT INTO tpcds.customer_address_ext1 SELECT * FROM tpcds.customer_address;
INSERT INTO tpcds.customer_address_ext2 SELECT * FROM tpcds.customer_address;
```

☐ **NOTE**

The design of OBS foreign tables does not allow exporting files to a non-empty path. However, in concurrent export scenarios, multiple files are exported to the same path, causing an error.

Assume that a user concurrently exports data from the same table to the same OBS foreign table, and that one SQL statement is executed to export data when another SQL statement is being executed and has not generated any file on the OBS server. In this case, certain data is overwritten although both SQL statements are successfully executed. Therefore, you are advised not to concurrently export data to the same OBS foreign table.

----End

Concurrently Exporting Tables

Use the two foreign tables to export tables from the database to two buckets in OBS.

Step 1 Log in to the OBS data server through the management console. On the OBS server, create the buckets **/input-data1** and **/input-data2** for storing data files, and create data directories **/input-data1/data** and **/input-data2/data**, respectively, in the two buckets.

Step 2 In GaussDB(DWS), create foreign tables **tpcds.customer_address_ext1** and **tpcds.customer_address_ext2** for the OBS server to receive exported data.

OBS and the database are in the same region. Tables to be exported are **tpcds.customer_address** and **tpcds.customer_demographics**.

Export information is set as follows:

- The source data file directories are **/input-data1/data/** and **/input-data2/data/**, so **location** of **tpcds.customer_address_ext1** and **tpcds.customer_address_ext2** is set to **obs://input-data1/data/** and **obs://input-data2/data/**, respectively.

Information about data formats is set based on the detailed data format parameters specified during data export from GaussDB(DWS). The parameter settings are as follows:

- **format** is set to **CSV**.
- **encoding** is set to **UTF-8**.
- **delimiter** is set to **E'\x08'**.
- Configure **encrypt**. Its default value is **off**.
- **access_key** is set to the AK you have obtained. (mandatory)
- **secret_access_key** is set to the SK you have obtained. (mandatory)

  📖 **NOTE**

  access_key and **secret_access_key** have been obtained during user creation. Replace the italic part with the actual keys.

Based on the preceding settings, the foreign table is created using the following statements:

```
CREATE FOREIGN TABLE tpcds.customer_address_ext1
(
ca_address_sk          integer          ,
ca_address_id          char(16)          ,
ca_street_number        char(10)            ,
ca_street_name          varchar(60)          ,
ca_street_type          char(15)            ,
ca_suite_number         char(10)            ,
ca_city            varchar(60)          ,
ca_county            varchar(30)          ,
ca_state            char(2)          ,
ca_zip            char(10)          ,
ca_country            varchar(20)          ,
ca_gmt_offset          decimal(5,2)            ,
ca_location_type         char(20)
)
SERVER gsmpp_server
OPTIONS(LOCATION 'obs://input-data1/data/',
FORMAT 'CSV',
ENCODING 'utf8',
DELIMITER E'\x08',
ENCRYPT 'off',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced'
)Write Only;
CREATE FOREIGN TABLE tpcds.customer_address_ext2
(
ca_address_sk          integer          ,
ca_address_id          char(16)          ,
ca_address_name          varchar(20)          ,
ca_address_code          integer          ,
```

```
ca_street_number          char(10)                ,
ca_street_name            varchar(60)             ,
ca_street_type            char(15)              ,
ca_suite_number           char(10)               ,
ca_city                 varchar(60)            ,
ca_county               varchar(30)            ,
ca_state                char(2)              ,
ca_zip                  char(10)             ,
ca_country              varchar(20)            ,
ca_gmt_offset           decimal(5,2)
)
SERVER gsmpp_server
OPTIONS(LOCATION 'obs://input_data2/data/',
FORMAT 'CSV',
ENCODING 'utf8',
DELIMITER E'\x08',
QUOTE E'\x1b',
ENCRYPT 'off',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced'
)Write Only;
```

Step 3 In GaussDB(DWS), export the data tables **tpcds.customer_address** and **tpcds.warehouse** in parallel to the foreign tables **tpcds.customer_address_ext1** and **tpcds.customer_address_ext2**, respectively.

```
INSERT INTO tpcds.customer_address_ext1 SELECT * FROM tpcds.customer_address;
INSERT INTO tpcds.customer_address_ext2 SELECT * FROM tpcds.warehouse;
```

----End

4.5.1.3 Exporting ORC Data to OBS

4.5.1.3.1 Planning Data Export

For details about exporting data to OBS, see **Planning Data Export**.

For details about the data types that can be exported to OBS, see **Table 4-6**.

For details about HDFS data export or MRS configuration, see the *MapReduce Service User Guide*.

4.5.1.3.2 Creating a Foreign Server

For details about creating a foreign server on OBS, see **Creating a Foreign Server**.

For details about creating a foreign server in HDFS, see **Manually Creating a Foreign Server**.

4.5.1.3.3 Creating a Foreign Table

After operations in **Creating a Foreign Server** are complete, create an OBS/HDFS write-only foreign table in the GaussDB(DWS) database to access data stored in OBS/HDFS. The foreign table is write-only and can be used only for data export.

The syntax for creating a foreign table is as follows. For details, see the syntax **CREATE FOREIGN TABLE (SQL on Hadoop or OBS)**.

```
CREATE FOREIGN TABLE [ IF NOT EXISTS ] table_name
( [ { column_name type_name
    [ { [CONSTRAINT constraint_name] NULL |
    [CONSTRAINT constraint_name] NOT NULL |
```

```
    column_constraint [...]} ] |
    table_constraint [, ...]} [, ...] ] )
SERVER dfs_server
OPTIONS ( { option_name ' value ' } [, ...] )
[ {WRITE ONLY }]
DISTRIBUTE BY {ROUNDROBIN | REPLICATION}
[ PARTITION BY ( column_name ) [ AUTOMAPPED ] ] ;
```

For example, when creating a foreign table named *product_info_ext_obs*, set
parameters in the syntax as follows:

- **table_name**

  Specifies the name of the foreign table to be created.

- **Table column definitions**

  - **column_name**: specifies the name of a column in the foreign table.

  - **type_name**: specifies the data type of the column.

  Multiple columns are separate by commas (,).

- **SERVER dfs_server**

  Specifies the foreign server name of the foreign table. This server must exist.
  The foreign table connects to OBS/HDFS to read data through the foreign
  server.

  Enter the name of the foreign server created by following steps in **Creating a
  Foreign Server**.

- **OPTIONS parameters**

  These are parameters associated with the foreign table. The key parameters
  are as follows:

  - **format**: specifies the format of the exported data file. The ORC format is
    supported.

  - **foldername**: (mandatory) specifies the data source file directory in the
    foreign table. OBS: specifies the OBS path of the data source file. You
    only need to enter */Bucket name/Folder directory level/*. HDFS: specifies
    the path in the HDFS file system. This parameter is mandatory for the
    write-only foreign table.

  - **encoding**: specifies the encoding of the data source file in the foreign
    table. The default value is **utf8**.

  - filesize

    Specifies the file size of a write-only foreign table, in MB. If this
    parameter is not specified, the file size in the distributed file system
    configuration is used by default. This syntax is available only for the
    write-only foreign table.

    Value range: an integer ranging from 1 to 1024

    📖 NOTE

      The **filesize** parameter is valid only for the ORC-formatted write-only HDFS
      foreign table.

  - compression

    (Optional) Specifies the compression mode of ORC files. This syntax is
    available only for the write-only foreign table.

    Value range: **zlib**, **snappy**, and **lz4** The default value is **snappy**.

– version

(Optional) Specifies the ORC version number. This syntax is available only for the write-only foreign table.

Value range: Only **0.12** is supported. The default value is **0.12**.

– dataencoding

(Optional) Specifies the data code of the data table to be exported when the database code is different from the data code of the data table. For example, the database code is Latin-1, but the data in the exported data table is in UTF-8 format. If this parameter is not specified, the database encoding format is used by default. This syntax is valid only for the write-only HDFS foreign table.

Value range: data code types supported by the database encoding

☐ NOTE

The **dataencoding** parameter is valid only for the ORC-formatted write-only HDFS foreign table.

- **Other parameters in the syntax**

  Other parameters are optional and can be configured as required. In this example, they do not need to be configured.

  Based on the preceding settings, the command for creating the foreign table is as follows:

```
DROP FOREIGN TABLE IF EXISTS product_info_ext_obs;

-- Create an OBS foreign table that does not contain partition columns. The foreign server associated
with the table is obs_server, the file format on OBS corresponding to the table is ORC, and the data
storage path on OBS is/mybucket/data/.

CREATE FOREIGN TABLE product_info_ext_obs
(
    product_price              integer        ,
    product_id                 char(30)       ,
    product_time               date           ,
    product_level              char(10)        ,
    product_name                 varchar(200)   ,
    product_type1                varchar(20)    ,
    product_type2                char(10)        ,
    product_monthly_sales_cnt    integer         ,
    product_comment_time         date          ,
    product_comment_num          integer         ,
    product_comment_content      varchar(200)
) SERVER obs_server
OPTIONS (
format 'orc',
foldername '/mybucket/demo.db/product_info_orc/',
  compression 'snappy',
    version '0.12'
) Write Only;
```

4.5.1.3.4 Exporting Data

Procedure

Step 1 Export data.

```
INSERT INTO [Foreign table name] SELECT * FROM [Source table name];
```

----End

Examples

- **Example 1**: Export data from table **product_info_output** to a data file using the **product_info_output_ext** foreign table.

  INSERT INTO *product_info_output_ext* SELECT * FROM *product_info_output*;

  If information similar to the following is displayed, the data has been exported.

  INSERT 0 10

- **Example 2**: Export part of the data to a data file by specifying the filter condition **WHERE** *product_price>500*.

  INSERT INTO *product_info_output_ext* SELECT * FROM *product_info_output* WHERE *product_price>500*;

📖 **NOTE**

Data of a special type, such as RAW, is exported as a binary file, which cannot be recognized by the import tool. As a result, you need to use the **RAWTOHEX()** function to convert it to the hexadecimal format before export.

4.5.2 Exporting ORC Data to MRS

4.5.2.1 Overview

GaussDB(DWS) allows you to export ORC data to MRS using an HDFS foreign table. You can specify the export mode and export data format in the foreign table. Data is exported from GaussDB(DWS) in parallel using multiple DNs and stored in HDFS. In this way, the overall export performance is improved.

- The CN only plans data export tasks and delivers the tasks to DNs. In this case, the CN is released to process external requests.

- The computing capability and bandwidth of all the DNs are fully leveraged to export data.

- Multiple HDFS servers can export data concurrently. The export path can be empty. The naming rule of the path must be the same as that of the exported file.

- MRS connects to GaussDB(DWS) cluster nodes. The export rate is affected by the network bandwidth.

- Data files in the ORC format are supported.

Naming Rules of Exported Files

The rules for naming ORC data files exported from GaussDB(DWS) are as follows:

1. Data exported to MRS (HDFS): When data is exported from a DN, the data is stored in HDFS in the segment format. The file is named in the format of **mpp**_*Database name_Schema name_Table name_Node name_n*.**orc**. *n* is a natural number starting from 0 in ascending order, for example, 0, 1, 2, 3.

2. You are advised to export data from different clusters or databases to different paths. The maximum size of an ORC file is 128 MB, and that of a stripe file is 64 MB.

3. After the export is complete, the **_SUCCESS** file is generated.

4.5.2.2 Planning Data Export

For details about the data types that can be exported to MRS, see **Table 4-6**.

For details about HDFS data export or MRS configuration, see the *MapReduce Service User Guide*.

4.5.2.3 Creating a Foreign Server

For details about creating a foreign server on HDFS, see **Manually Creating a Foreign Server**.

4.5.2.4 Creating a Foreign Table

After operations in **Creating a Foreign Server** are complete, create an HDFS write-only foreign table in the GaussDB(DWS) database to access data stored in HDFS. The foreign table is write-only and can be used only for data export.

The syntax for creating a foreign table is as follows. For details, see **CREATE FOREIGN TABLE (SQL on Hadoop or OBS)**.

```
CREATE FOREIGN TABLE [ IF NOT EXISTS ] table_name
( [ { column_name type_name
  [ { [CONSTRAINT constraint_name] NULL |
  [CONSTRAINT constraint_name] NOT NULL |
   column_constraint [...]} ] |
   table_constraint [, ...]} [, ...] ] )
 SERVER dfs_server
 OPTIONS ( { option_name ' value ' } [, ...] )
 [ {WRITE ONLY }]
 DISTRIBUTE BY {ROUNDROBIN | REPLICATION}
 [ PARTITION BY ( column_name ) [ AUTOMAPPED ] ] ;
```

For example, when creating a foreign table *product_info_ext_obs*, configure the parameters in the syntax as follows.

- **table_name**

  Specifies the name of the foreign table.

- **Table column definitions**

  – **column_name**: specifies the name of a column in the foreign table.

  – **type_name**: specifies the data type of the column.

  Multiple columns are separate by commas (,).

- **SERVER dfs_server**

  Specifies the foreign server name of the foreign table. This server must exist. The foreign table connects to OBS/HDFS to read data through the foreign server.

  Enter the name of the foreign server created in **Creating a Foreign Server**.

- **OPTIONS parameters**

  These parameters are associated with the foreign table. The key parameters are as follows:

  – **format**: specifies the format of the exported data file. The ORC format is supported.

  – **foldername**: specifies the directory of the data source file in the foreign table, that is, the corresponding file directory in HDFS. This parameter is

mandatory for write-only foreign tables and optional for read-only foreign tables.

- **encoding**: specifies the encoding format of the data source file in the foreign table. The default value is **utf8**.

- filesize

  (Optional) Specifies the file size of a write-only foreign table. If this parameter is not specified, the file size in the distributed file system is used by default. This syntax is available only for the write-only foreign table.

  Value range: an integer ranging from 1 to 1024

  > **NOTE**
  >
  > The **filesize** parameter is valid only for the write-only HDFS foreign table in ORC format.

- compression

  (Optional) Specifies the compression mode of ORC files. This syntax is available only for the write-only foreign table.

  Value range: **zlib**, **snappy**, and **lz4**. The default value is **snappy**.

- version

  (Optional) Specifies the ORC version number. This syntax is available only for the write-only foreign table.

  Value range: Only **0.12** is supported. The default value is **0.12**.

- dataencoding

  (Optional) Specifies the data encoding of the data table to be exported when the database encoding is different from the data encoding of the data table. For example, the database encoding is Latin-1, but the data encoding of the exported data table is in UTF-8 format. If this parameter is not specified, the database encoding is used by default. This syntax is valid only for the write-only HDFS foreign table.

  Value range: data encoding types supported by the database encoding

  > **NOTE**
  >
  > The **dataencoding** parameter is valid only for the write-only HDFS foreign table in ORC format.

- **Other parameters in the syntax**

  Other parameters are optional and can be configured as required. In this example, they do not need to be configured. For details, see **CREATE FOREIGN TABLE (SQL on Hadoop or OBS)**.

  Based on the preceding settings, the command for creating the foreign table is as follows:

```
DROP FOREIGN TABLE IF EXISTS product_info_ext_obs;

-- Create an OBS foreign table that does not contain partition columns. The foreign server associated with the table is hdfs_server, the format of the file on HDFS corresponding to the table is ORC, and the data storage path on OBS is /user/hive/warehouse/product_info_orc/.

CREATE FOREIGN TABLE product_info_ext_obs
(
    product_price           integer       ,
    product_id              char(30)      ,
```

```
        product_time            date        ,
        product_level           char(10)      ,
        product_name             varchar(200)   ,
        product_type1           varchar(20)    ,
        product_type2           char(10)       ,
        product_monthly_sales_cnt  integer       ,
        product_comment_time      date        ,
        product_comment_num        integer       ,
        product_comment_content     varchar(200)
) SERVER obs_server
OPTIONS (
format 'orc',
foldername '/user/hive/warehouse/product_info_orc/',
  compression 'snappy',
    version '0.12'
) Write Only;
```

4.5.2.5 Exporting Data

Procedure

Step 1 Export data.

```
INSERT INTO [Foreign table name] SELECT * FROM [Source table name];
```

----End

Examples

- **Example 1**: Export data from table **product_info_output** to a data file using the **product_info_output_ext** foreign table.

  ```
  INSERT INTO product_info_output_ext SELECT * FROM product_info_output;
  ```

  If information similar to the following is displayed, the data has been exported.

  ```
  INSERT 0 10
  ```

- **Example 2**: Export part of the data to a data file by specifying the filter condition **WHERE** *product_price>500*.

  ```
  INSERT INTO product_info_output_ext SELECT * FROM product_info_output WHERE product_price>500;
  ```

  📖 **NOTE**

  Data of a special type, such as RAW, is exported as a binary file, which cannot be recognized by the import tool. As a result, you need to use the **RAWTOHEX()** function to convert it to the hexadecimal format before export.

4.5.3 Using GDS to Export Data to a Remote Server

4.5.3.1 Exporting Data In Parallel Using GDS

In high-concurrency scenarios, you can use GDS to export data from a database to a common file system.

In the current GDS version, data can be exported from a database to a pipe file.

- When the local disk space of the GDS user is insufficient:
  - The data exported from GDS is compressed using the pipe to occupy less disk space.
  - The exported data is transferred through the pipe to the HDFS server for storage.

- If you need to cleanse data before exporting data:
  – You can compile programs as needed and read streaming data from pipes in real time.

📖 **NOTE**

- The current version does not support data export through GDS in SSL mode. Do not use GDS in SSL mode.
- All pipe files mentioned in this section refer to named pipes on Linux.

Overview

Using foreign tables: A GDS foreign table specifies the exported file format and export mode. Data is exported in parallel through multiple DNs from the database to data files, which improves the overall data export performance. The data files cannot be directly exported to HDFS.

- The CN only plans data export tasks and delivers the tasks to DNs. In this case, the CN is released to process other tasks.
- In this way, the computing capabilities and bandwidths of all the DNs are fully leveraged to export data.

Figure 4-11 Exporting data using foreign tables
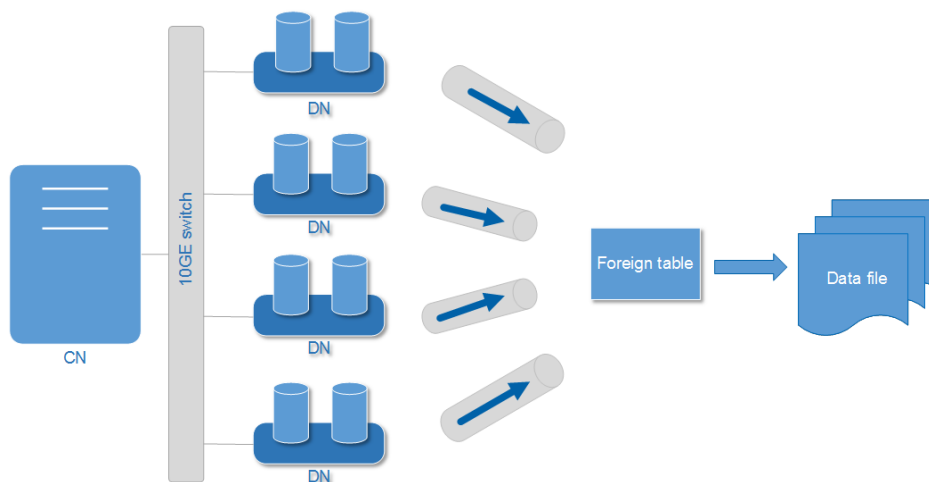
Related Concepts

- **Data file**: A TEXT, CSV, or FIXED file that stores data exported from the GaussDB(DWS) database.
- **Foreign table**: A table that stores information, such as the format, location, and encoding format of a data file.
- **GDS**: A data service tool. To export data, deploy it on the server where data files are stored.
- **Table**: Tables in the database, including row-store tables and column-store tables. Data in the data files is exported from these tables.
- **Remote mode**: Service data in a cluster is exported to hosts outside the cluster.

Exporting a Schema

Data can be exported to GaussDB(DWS) in **Remote** mode.

- **Remote mode**: Service data in a cluster is exported to hosts outside the cluster.
    - In this mode, multiple GDSs are used to concurrently export data. One GDS can export data for only one cluster at a time.
    - The data export rate of a GDS that resides on the same intranet as cluster nodes is limited by the network bandwidth. A 10GE configuration is recommended.
    - Data files in TEXT or CSV format are supported. The size of data in a single row must be less than 1 GB.

Data Export Process

Figure 4-12 Concurrent data export
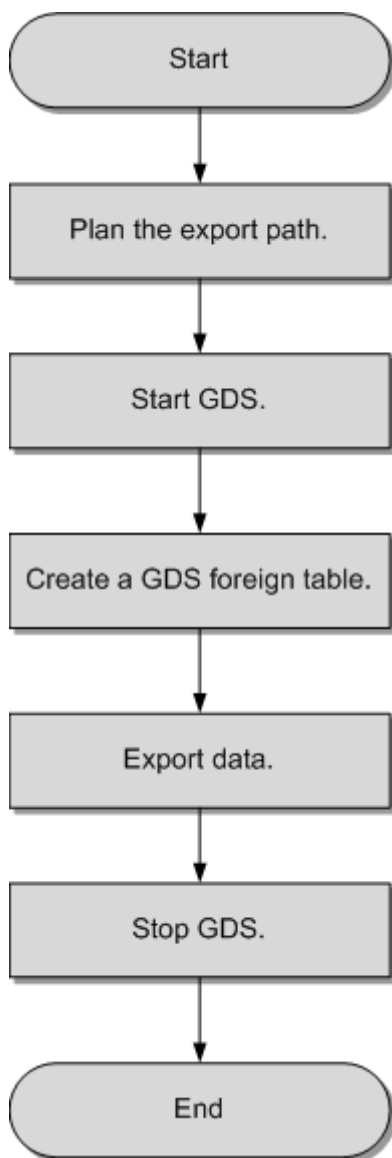
Table 4-23 Process description

Process	Description	Subtask
Plan data export.	Prepare data to be exported and plan the export path for the mode to be selected. For details, see **Planning Data Export**.	-
Start GDS.	If the **Remote** mode is selected, install, configure, and start GDS on data servers. For details, see **Installing, Configuring, and Starting GDS**.	-
Create a foreign table,	Create a foreign table to help GDS specify information about a data file. The foreign table stores information, such as the location, format, encoding, and inter-data delimiter of a data file. For details, see **Creating a GDS Foreign Table**.	-
Export data.	After the foreign table is created, run the **INSERT** statement to efficiently export data to data files. For details, see **Exporting Data**.	-
Stop GDS.	Stop GDS after data is exported. For details, see **Stopping GDS**.	-

4.5.3.2 Planning Data Export

Scenarios

Before you use GDS to export data from a cluster, prepare data to be exported and plan the export path.

Planning an Export Path

- **Remote** mode

Step 1 Log in to the GDS data server as user **root** and create the **/output_data** directory for storing data files.

mkdir -p */output_data*

Step 2 (Optional) Create a user and the user group to which it belongs. This user is used to start GDS and must have the write permission on the directory for storing data files.

```
groupadd gdsgrp
useradd -g gdsgrp gdsuser
```

If the following information is displayed, the user and user group already exist. Skip this step.

```
useradd: Account 'gdsuser' already exists.
groupadd: Group 'gdsgrp' already exists.
```

Step 3 Change the directory owner to **gdsuser**.

```
chown -R gdsuser:gdsgrp /output_data
```

----End

4.5.3.3 Installing, Configuring, and Starting GDS

GDS is a data service tool provided by GaussDB(DWS). Using the foreign table mechanism, this tool helps export data at a high speed.

For details, see **Installing, Configuring, and Starting GDS**.

4.5.3.4 Creating a GDS Foreign Table

Procedure

Step 1 Set the **location** parameter for the foreign table based on the path planned in **Planning Data Export**.

- **Remote** mode

  Set the **location** parameter to the URL of the directory that stores the data files.

  – You do not need to specify a file name in the URL.

  – If multiple URLs are configured, only the first URL is effective.

  For example:

  The IP address of the GDS data server is 192.168.0.90. The listening port number set during GDS startup is 5000. The directory for storing data files is **/output_data**.

  In this case, set the **location** parameter to **gsfs://192.168.0.90:5000/**.

  📖 NOTE

  – **location** can be set to a subdirectory, for example, **gsfs://192.168.0.90:5000/2019/11/**, so that the same table can be exported to different directories by date.

  – In the current version, when an **export** task is executed, the system checks whether the **/output_data/2019/11** directory exists. If the directory does not exist, the system creates it. During the export, files are written to this directory. In this way, you do not need to manually run the **mkdir -p /output_data/2019/11** command after creating or modifying a foreign table.

Step 2 Set data format parameters in the foreign table based on the planned data file formats. For details about format parameters, see data format parameters.

Step 3 Create a GDS foreign table based on the parameter settings in the preceding steps. For details about how to create a foreign table, see CREATE FOREIGN TABLE (for GDS Import and Export).

----End

Example

- **Example**: Create the GDS foreign table **foreign_tpcds_reasons** for the source data. Data is to be exported as CSV files.

  Data export mode settings are as follows:

  The data server resides on the same intranet as the cluster. The IP address of the data server is 192.168.0.90. Data is to be exported as CSV files. The **Remote** mode is selected for parallel data export.

  Assume that the directory for storing data files is **/output_data/** and the GDS listening port is 5000 when GDS is started. Therefore, the **location** parameter is set to **gsfs://192.168.0.90:5000/**.

  Data format parameter settings are as follows:

  - **format** is set to **CSV**.

  - **encoding** is set to **UTF-8**.

  - **delimiter** is set to **E'\x08'**.

  - **quote** is set to **E'\x1b'**.

  - **null** is set to an empty string without quotation marks.

  - **escape** is set to the same value as that of **quote** by default.

  - **header** is set to **false**, indicating that the first row is identified as a data row in an exported file.

  - **EOL** is set to **0X0A**.

  The foreign table is created using the following statement:

```
CREATE FOREIGN TABLE foreign_tpcds_reasons
(
  r_reason_sk    integer       not null,
  r_reason_id    char(16)      not null,
  r_reason_desc  char(100)
)
SERVER gsmpp_server
OPTIONS (LOCATION 'gsfs://192.168.0.90:5000/',
FORMAT 'CSV',
DELIMITER E'\x08',
QUOTE E'\x1b',
NULL '',
EOL '0x0a'
)
WRITE ONLY;
```

4.5.3.5 Exporting Data

Prerequisites

Ensure that the IP addresses and ports of servers where CNs and DNs are deployed can connect to those of the GDS server.

Procedure

Step 1 Export data.

INSERT INTO *[Foreign table name]* **SELECT * FROM** *[Source table name]*;

> 📖 **NOTE**
>
> Create batch processing scripts to export data in parallel. The degree of parallelism depends on the server resource usage. You can test several tables and monitor resource usage to determine whether to increase or reduce the amount. Common resource monitoring commands include **top** for memory and CPU usage, **iostat** for I/O usage, and **sar** for networks. For details about application cases, see **Exporting Data Using Multiple Threads**.

----End

Examples

- **Example 1**: Export data from the **reason** table to data files through the **foreign_tpcds_reasons** foreign table.

  INSERT INTO foreign_tpcds_reasons SELECT * FROM tpcds.reason;

- **Example 2**: Export part of the data to data files by specifying the filter condition **r_reason_sk =1**.

  INSERT INTO foreign_tpcds_reasons SELECT * FROM tpcds.reason WHERE r_reason_sk=1;

- **Example 3**: Data of a special type, such as RAW, is exported as a binary file, which cannot be recognized by the import tool. You need to use the RAWTOHEX() function to convert it to hexadecimal the format before export.

  INSERT INTO foreign_tpcds_reasons SELECT RAWTOHEX(c) FROM tpcds.reason;

4.5.3.6 Stopping GDS

GDS is a data service tool provided by GaussDB(DWS). Using the foreign table mechanism, this tool helps export data at a high speed.

For details, see **Stopping GDS**.

4.5.3.7 Examples of Exporting Data Using GDS

Exporting Data in Remote Mode

The data server and the cluster reside on the same intranet, the IP address of the data server is **192.168.0.90**, and data source files are in CSV format. In this scenario, data is exported in parallel in **Remote** mode.

To export data in parallel in **Remote** mode, perform the following operations:

1. Log in to the GDS data server as user **root**, create the **/output_data** directory for storing data files, and create user **gds_user** and its user group.

   mkdir -p */output_data*

2. (Optional) Create a user and the user group it belongs to. The user is used to start GDS. If the user and user group exist, skip this step.

   groupadd gdsgrp
   useradd -g gdsgrp gds_user

3. Change the owner of the **/output_data** directory on the data server to **gds_user**.

   chown -R *gds_user:gdsgrp /output_data*

4. Log in to the data server as user **gds_user** and start GDS.

The GDS installation path is **/opt/bin/dws/gds**. Exported data files are stored in **/output_data/**. The IP address of the data server is **192.168.0.90**. The GDS listening port is **5000**. GDS runs in daemon mode.

```
/opt/bin/dws/gds/bin/gds -d /output_data -p 192.168.0.90:5000 -H 10.10.0.1/24 -D
```

5. In the database, create the foreign table **foreign_tpcds_reasons** for receiving data from the data server.

Data export mode settings are as follows:

– The directory for storing exported files is **/output_data/** and the GDS listening port is **5000** when GDS is started. The directory created for storing exported files is **/output_data/**. Therefore, the **location** parameter is set to **gsfs://192.168.0.90:5000/**.

Data format parameter settings are as follows:

– **format** is set to **CSV**.

– **encoding** is set to **UTF-8**.

– **delimiter** is set to **E'\x08'**.

– **quote** is set to **E'\x1b'**.

– **null** is set to an empty string without quotation marks.

– **escape** defaults to the value of **quote**.

– **header** is set to **false**, indicating that the first row is identified as a data row in an exported file.

Based on the above settings, the foreign table is created using the following statement:

```
CREATE FOREIGN TABLE foreign_tpcds_reasons
(
  r_reason_sk    integer       not null,
  r_reason_id    char(16)      not null,
  r_reason_desc  char(100)
)
SERVER gsmpp_server
OPTIONS
(
LOCATION 'gsfs://192.168.0.90:5000/',
FORMAT 'CSV',
ENCODING 'utf8',
DELIMITER E'\x08',
QUOTE E'\x1b',
NULL ''
)
WRITE ONLY;
```

6. In the database, export data to data files through the foreign table **foreign_tpcds_reasons**.

```
INSERT INTO foreign_tpcds_reasons SELECT * FROM tpcds.reason;
```

7. After data export is complete, log in to the data server as user **gds_user** and stop GDS.

The GDS process ID is **128954**.

```
ps -ef|grep gds
gds_user 128954      1  0 15:03 ?        00:00:00 gds -d /output_data -p 192.168.0.90:5000 -D
gds_user 129003 118723  0 15:04 pts/0    00:00:00 grep gds
kill -9 128954
```

Exporting Data Using Multiple Threads

The data server and the cluster reside on the same intranet, the IP address of the data server is **192.168.0.90**, and data source files are in CSV format. In this

scenario, data is concurrently exported to two target tables using multiple threads in **Remote** mode.

To concurrently export data using multiple threads in **Remote** mode, perform the following operations:

1. Log in to the GDS data server as user **root**, create the **/output_data** directory for storing data files, and create the database user and its user group.
   mkdir -p */output_data*
   groupadd gdsgrp
   useradd -g *gdsgrp gds_user*

2. Change the owner of the **/output_data** directory on the data server to **gds_user**.
   chown -R gds_user:gdsgrp */output_data*

3. Log in to the data server as user **gds_user** and start GDS.

   The GDS installation path is **/opt/bin/dws/gds**. Exported data files are stored in **/output_data/**. The IP address of the data server is **192.168.0.90**. The GDS listening port is **5000**. GDS runs in daemon mode. The degree of parallelism is 2.
   /opt/bin/dws/gds/bin/gds -d */output_data* **-p** *192.168.0.90:5000* **-H** 10.10.0.1/24 **-D -t** 2

4. In GaussDB(DWS), create the foreign tables **foreign_tpcds_reasons1** and **foreign_tpcds_reasons2** for receiving data from the data server.
   - Data export mode settings are as follows:

     - The directory for storing exported files is **/output_data/** and the GDS listening port is **5000** when GDS is started. The directory created for storing exported files is **/output_data/**. Therefore, the **location** parameter is set to **gsfs://192.168.0.90:5000/**.

   - Data format parameter settings are as follows:

     - **format** is set to **CSV**.

     - **encoding** is set to **UTF-8**.

     - **delimiter** is set to **E'\x08'**.

     - **quote** is set to **E'\x1b'**.

     - **null** is set to an empty string without quotation marks.

     - **escape** defaults to the value of **quote**.

     - **header** is set to **false**, indicating that the first row is identified as a data row in an exported file.

   Based on the preceding settings, the foreign table **foreign_tpcds_reasons1** is created using the following statement:

```
CREATE FOREIGN TABLE foreign_tpcds_reasons1
(
  r_reason_sk    integer     not null,
  r_reason_id    char(16)    not null,
  r_reason_desc  char(100)
)
SERVER gsmpp_server
OPTIONS
(
LOCATION 'gsfs://192.168.0.90:5000/',
FORMAT 'CSV',
```

```
ENCODING 'utf8',
DELIMITER E'\x08',
QUOTE E'\x1b',
NULL ''
)
WRITE ONLY;
```

Based on the preceding settings, the foreign table **foreign_tpcds_reasons2** is created using the following statement:

```
CREATE FOREIGN TABLE foreign_tpcds_reasons2
(
  r_reason_sk     integer     not null,
  r_reason_id     char(16)    not null,
  r_reason_desc   char(100)
)
SERVER gsmpp_server
OPTIONS
(
LOCATION 'gsfs://192.168.0.90:5000/',
FORMAT 'CSV',
DELIMITER E'\x08',
QUOTE E'\x1b',
NULL ''
)
WRITE ONLY;
```

5. In the database, export data from table **reasons1** through the foreign table **foreign_tpcds_reasons1** and from table **reasons2** through the foreign table **foreign_tpcds_reasons2** to **/output_data**.
   ```
   INSERT INTO foreign_tpcds_reasons1 SELECT * FROM tpcds.reason;
   INSERT INTO foreign_tpcds_reasons2 SELECT * FROM tpcds.reason;
   ```

6. After data export is complete, log in to the data server as user **gds_user** and stop GDS.

   The GDS process ID is **128954**.
   ```
   ps -ef|grep gds
   gds_user 128954      1  0 15:03 ?        00:00:00 gds -d /output_data -p 192.168.0.90:5000 -D -t 2
   gds_user 129003 118723  0 15:04 pts/0    00:00:00 grep gds
   kill -9 128954
   ```

Exporting Data Through a Pipe

Step 1 Start GDS.
```
gds -d /***/gds_data/ -D -p 192.168.0.1:7789 -l /***/gds_log/aa.log -H 0/0 -t 10 -D
```

If you need to set the timeout interval of a pipe, use the **--pipe-timeout** parameter.

Step 2 Export data.

1. Log in to the database, create an internal table, and write data to the table.
   ```
   CREATE TABLE test_pipe( id integer not null, sex text not null, name text ) ;


   INSERT INTO test_pipe values(1,2,'11111111111111');
   INSERT INTO test_pipe values(2,2,'11111111111111');
   INSERT INTO test_pipe values(3,2,'11111111111111');
   INSERT INTO test_pipe values(4,2,'11111111111111');
   INSERT 0 1
   ```

2. Create a write-only foreign table.
   ```
   CREATE FOREIGN TABLE foreign_test_pipe_tw( id integer not null, age text not null, name  text )
   SERVER gsmpp_server OPTIONS (LOCATION 'gsfs://192.168.0.1:7789/', FORMAT 'text', DELIMITER ',',
   NULL '', EOL '0x0a' ,file_type 'pipe', auto_create_pipe 'false') WRITE ONLY;
   ```

3. Execute the export statement. In this case, the statements are blocked.
   ```
   INSERT INTO foreign_test_pipe_tw select * from test_pipe;
   ```

Step 3 Export data through the GDS pipes.

1. Log in to GDS and go to the GDS data directory.
   ```
   cd /***/gds_data/
   ```

2. Create a pipe. If **auto_create_pipe** is set to **true**, skip this step.
   ```
   mkfifo postgres_public_foreign_test_pipe_tw.pipe
   ```

   ## NOTE

   A pipe will be automatically cleared after an operation is complete. To perform another operation, create a pipe file again.

3. Read data from the pipe and write it to a new file.
   ```
   cat postgres_public_foreign_test_pipe_tw.pipe > postgres_public_foreign_test_pipe_tw.txt
   ```

4. To compress the exported files, run the following command:
   ```
   gzip -9 -c < postgres_public_foreign_test_pipe_tw.pipe  > out.gz
   ```

5. To export the content from the pipe to the HDFS server, run the following command:
   ```
   cat postgres_public_foreign_test_pipe_tw.pipe  | hdfs dfs -put -  /user/hive/***/test_pipe.txt
   ```

Step 4 Verify the exported data.

1. Check whether the exported file is correct.
   ```
   cat postgres_public_foreign_test_pipe_tw.txt
   3,2,11111111111111
   1,2,11111111111111
   2,2,11111111111111
   4,2,11111111111111
   ```

2. View the compressed file.
   ```
   vim out.gz
   3,2,11111111111111
   1,2,11111111111111
   2,2,11111111111111
   4,2,11111111111111
   ```

3. View the data exported to the HDFS server.
   ```
   hdfs dfs -cat /user/hive/***/test_pipe.txt
   3,2,11111111111111
   1,2,11111111111111
   2,2,11111111111111
   4,2,11111111111111
   ```

----End

Exporting Data Through Multi-Process Pipes

GDS also supports importing and exporting data through multi-process pipes. That is, one foreign table corresponds to multiple GDSs.

The following takes exporting a local file as an example.

Step 1 Start multiple GDSs.
```
gds -d /***/gds_data/ -D -p 192.168.0.1:7789 -l /***/gds_log/aa.log -H 0/0 -t 10 -D
gds -d /***/gds_data_1/ -D -p 192.168.0.1:7790 -l /***/gds_log/aa.log -H 0/0 -t 10 -D
```

If you need to set the timeout interval of a pipe, use the **--pipe-timeout** parameter.

Step 2 Export data.

1. Log in to the database and create an internal table.
   ```
   CREATE TABLE test_pipe (id integer not null, sex text not null, name  text);
   ```

2. Write data.
    ```
    INSERT INTO test_pipe values(1,2,'11111111111111');
    INSERT INTO test_pipe values(2,2,'11111111111111');
    INSERT INTO test_pipe values(3,2,'11111111111111');
    INSERT INTO test_pipe values(4,2,'11111111111111');
    ```

3. Create a write-only foreign table.
    ```
    CREATE FOREIGN TABLE foreign_test_pipe_tw( id integer not null, age text not null, name  text )
    SERVER gsmpp_server OPTIONS (LOCATION 'gsfs://192.168.0.1:7789|gsfs://192.168.0.1:7790/',
    FORMAT 'text', DELIMITER ',',  NULL '', EOL '0x0a' ,file_type 'pipe', auto_create_pipe 'false') WRITE
    ONLY;
    ```

4. Execute the export statement. In this case, the statements are blocked.
    ```
    INSERT INTO foreign_test_pipe_tw select * from test_pipe;
    ```

Step 3 Export data through the GDS pipes.

1. Log in to GDS and go to each GDS data directory.
    ```
    cd /***/gds_data/
    cd /***/gds_data_1/
    ```

2. Create a pipe. If **auto_create_pipe** is set to **true**, skip this step.
    ```
    mkfifo postgres_public_foreign_test_pipe_tw.pipe
    ```

3. Read each pipe and write the new file to the pipes.
    ```
    cat postgres_public_foreign_test_pipe_tw.pipe > postgres_public_foreign_test_pipe_tw.txt
    ```

Step 4 Verify the exported data.
```
cat /***/gds_data/postgres_public_foreign_test_pipe_tw.txt
3,2,11111111111111
cat /***/gds_data_1/postgres_public_foreign_test_pipe_tw.txt
1,2,11111111111111
2,2,11111111111111
4,2,11111111111111
```

----End

4.6 Other Operations

4.6.1 GDS Pipe FAQs

Precautions

- GDS supports concurrent import and export. The **gds -t** parameter is used to set the size of the thread pool and control the maximum number of concurrent working threads. But it does not accelerate a single SQL task. The default value of **gds -t** is **8**, and the upper limit is **200**. When using the pipe function to import and export data, ensure that the value of **-t** is greater than or equal to the number of concurrent services. In the dual-cluster interconnection scenario, the value of **-t** must be greater than or equal to twice the number of concurrent services.

- Data in pipes is deleted once read. Therefore, ensure that no other program except GDS reads data in the pipe during import or export. Otherwise, data may be lost, task errors may occur, or the exported files may be disordered.

- Concurrent import and export of foreign tables with the same location are not supported. That is, multiple threads of GDS cannot read or write pipe files at the same time.

- A single import or export task of GDS identifies only one pipe. Therefore, do not carry wildcard characters ({}[]?) in the location address set for the GDS foreign table. Example:

  CREATE FOREIGN TABLE foreign_test_pipe_tr(like test_pipe) SERVER gsmpp_server OPTIONS (LOCATION 'gsfs://192.168.0.1:7789/foreign_test_*', FORMAT 'text', DELIMITER ',', NULL '', EOL '0x0a' ,file_type 'pipe',auto_create_pipe 'false');

- When the **-r** recursion parameter is enabled for GDS, only one pipe can be identified. That is, GDS identifies only one pipe in the current data directory and does not recursively search for it. Therefore, the **-r** parameter does not take effect in the pipe import and export scenarios.

- CN retry is not supported during the import and export through a pipe, because GDS cannot control the operations performed by peer users and programs on pipes.

- During the import, if the peer program does not write data into the pipe for more than one hour, the import task times out and an error is reported.

- During the export, if the peer program does not read data from the pipe for more than one hour by default, the export task times out and an error is reported.

- Ensure that the GDS version and kernel version support the function of importing and exporting data through pipes.

- If the **auto_create_pipe parameter** of the foreign table is set to **true**, a delay may occur when GDS automatically creates a pipe. Before any operation on a pipe, check whether the automatically created pipe exists and whether it is a pipe file.

- Once an import or export task through a GDS pipe is complete, the pipe is automatically deleted. However, the pipe deletion is delayed, if you manually terminate an import or export task. In this situation, the pipe is deleted after the timeout interval expires.

Common Troubleshooting Methods:

- Issue 1: **"/***/postgres_public_foreign_test_pipe_tr.pipe" must be named pipe.**

  Locating method: The type of the GDS foreign table **file_type** is pipe, but the operated file is a common file. Check whether the **postgres_public_foreign_test_pipe_tr.pipe** file is a pipe file.

- Issue 2: **could not open pipe "/***/ postgres_public_foreign_test_pipe_tw.pipe" cause by Permission denied**.

  Locating method: GDS does not have the permission to open the pipe file.

- Issue 3: **could not open source file /*****/ postgres_public_foreign_test_pipe_tw.pipe because timeout 300s for WRITING**.

  Locating method: Opening the pipe times out when GDS is used to export data. This is because the pipe is not created within 300 seconds after **auto_create_pipe** is set to **false**, or the pipe is created but is not read by any program within 300 seconds.

- Issue 4: **could not open source file /*****/ postgres_public_foreign_test_pipe_tw.pipe because timeout 300s for READING**.

Locating method: Opening the pipe times out when GDS is used to export data. This is because the pipe is not created within 300 seconds after **auto_create_pipe** is set to **false**, or the pipe is created but is not written by any program within 300 seconds.

- Issue 5: **could not poll writing source pipe file "/****/ postgres_public_foreign_test_pipe_tw.pipe" timeout 300s.**

  Locating method: If the GDS does not receive any write event on the pipe within 300 seconds during data export, the pipe is not read for more than 300 seconds.

- Issue 6: **could not poll reading source pipe file "/****/ postgres_public_foreign_test_pipe_tw.pipe" timeout 300s.**

  Locating method: If the GDS does not receive any read event on the pipe within 300 seconds during data import, the pipe is not written for more than 300 seconds.

- Issue 7: **could not open pipe file "/***/ postgres_public_foreign_test_pipe_tw.pipe" for "WRITING" with error No such device or address**.

  Locating method: It indicates that the **/***/ postgres_public_foreign_test_pipe_tw.pipe** file is not read by any program. As a result, GDS cannot open the pipe file by writing.

4.6.2 Checking for Data Skew

Scenarios

Data skew causes the query performance to deteriorate. Before importing all the data from a table consisting of over 10 million records, you are advised to import some of the data and check whether data skew occurs and whether the distribution keys need to be changed. Troubleshoot the problems if any. It is costly to address data skew and change the distribution keys after a large amount of data has been imported.

Context

GaussDB(DWS) uses a massively parallel processing (MPP) system of the shared-nothing architecture. The MPP performs horizontal partitioning to store tuples in service data tables on all DNs using proper distribution policies.

The following user table distribution policies are supported:

- Replication: stores a full table on each DN. You are advised to use the replication mode for tables containing a small volume of data.

- Hash: A distribution key must be specified for a user table. When a record is inserted, the system hashes it based on the distribution key and then stores it on the corresponding DN. You are advised to use the hash distribution policy for tables with a large volume of data.

- Round-robin: Each row in the table is sent to each DN in turn. Therefore, data is evenly distributed on each DN. If no proper distribution column can be found in a table with a large amount of data in hash mode, you are advised to use the round-robin distribution policy.

If an inappropriate distribution key is used, data skew may occur when you use the hash policy. Check for data skew when you use the hash distribution policy so that data can be evenly distributed to each DN. You are advised to use the column with few replicated values as the distribution key.

Procedure

Step 1 Analyze data source features and select candidate distribution columns that have more distinct values and evenly distributed data.

Step 2 Select a candidate column from **Step 1** to create a target table.
```
CREATE [ [ GLOBAL | LOCAL ] { TEMPORARY | TEMP } | UNLOGGED ] TABLE [ IF NOT EXISTS ] table_name
  ({ column_name data_type [ compress_mode ] [ COLLATE collation ] [ column_constraint [ ... ] ]
  | table_constraint   | LIKE source_table [ like_option [...] ] ] }
  [, ... ])   [ WITH ( {storage_parameter = value} [, ... ] ) ]
  [ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
  [ COMPRESS | NOCOMPRESS ]    [ TABLESPACE tablespace_name ]
  [ DISTRIBUTE BY { REPLICATION
             | ROUNDROBIN
             | { HASH ( column_name [,...] ) } } ];
```

Step 3 Import a small batch of data to the target table.

When importing a single data file, you can evenly split this file and import a part of it to check for the data skew in the target table.

Step 4 Check for data skew. (Replace *table_name* with the actual table name.)
```
SELECT a.count,b.node_name FROM (SELECT count(*) AS count,xc_node_id FROM table_name GROUP
BY xc_node_id) a, pgxc_node b WHERE a.xc_node_id=b.node_id ORDER BY a.count desc;
```

Step 5 If the data distribution deviation is less than 10% across DNs, data is evenly distributed and an appropriate distribution key has been selected. Delete the small batch of imported data and import full data to complete data migration.

If data distribution deviation across DNs is greater than or equal to 10%, data skew occurs. Remove this distribution key from the candidates in **Step 1**, delete the target table, and repeat **Step 2** through **Step 5**.

> **NOTE**
>
> The data distribution deviation indicates the difference between the actual data volume on DNs and the average data volume on DNs.

Step 6 (Optional) If you fail to select an appropriate distribution key after performing the preceding steps, select multiple columns from the candidates as distribution keys.

----End

Examples

Assume you want to select an appropriate distribution key for the **staffs** table.

1. Analyze the source data for the **staffs** table and select the **staff_ID**, **FIRST_NAME**, and **LAST_NAME** columns as candidate distribution keys.

2. Select the **staff_ID** column as the distribution key and create the target table **staffs**.
```
CREATE TABLE staffs
(
 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(2,2),
MANAGER_ID     NUMBER(6),
section_ID     NUMBER(4)
)
DISTRIBUTE BY hash(staff_ID);
```

3. Import a small batch of data to the target table **staffs**.

   There are eight DNs in the cluster based on the following query, and you are advised to import 80,000 records.
   ```
   SELECT count(*) FROM pgxc_node where node_type='D';
    count
   -------
       8
   (1 row)
   ```

4. Verify the data skew of the target table **staffs** whose distribution key is **staff_ID**:
   ```
   SELECT a.count,b.node_name FROM (select count(*) as count,xc_node_id FROM staffs GROUP BY
   xc_node_id) a, pgxc_node b WHERE a.xc_node_id=b.node_id ORDER BY a.count desc;
   count | node_name
   ------+-----------
   11010 | datanode4
   10000 | datanode3
   12001 | datanode2
    8995 | datanode1
   10000 | datanode5
    7999 | datanode6
    9995 | datanode7
   10000 | datanode8
   (8 rows)
   ```

5. The preceding query result indicates that the distribution deviation across DNs is greater than 10%. The data skew occurs. Therefore, delete **staff_ID** from the distribution key candidates and delete the **staffs** table.
   ```
   DROP TABLE staffs;
   ```

6. Use **staff_ID**, **FIRST_NAME**, and **LAST_NAME** as distribution keys and create the target table **staffs**.
   ```
   CREATE TABLE staffs
   (
   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(2,2),
   MANAGER_ID     NUMBER(6),
   section_ID     NUMBER(4)
   )
   DISTRIBUTE BY hash(staff_ID,FIRST_NAME,LAST_NAME);
   ```

7. Verify the data skew of the target table **staffs** whose distribution keys are **staff_ID**, **FIRST_NAME**, and **LAST_NAME**.
   ```
   SELECT a.count,b.node_name FROM (select count(*) as count,xc_node_id FROM staffs GROUP BY
   xc_node_id) a, pgxc_node b WHERE a.xc_node_id=b.node_id ORDER BY a.count desc;
   count | node_name
   ------+-----------
   10010 | datanode4
   10000 | datanode3
   ```

```
10001 | datanode2
 9995 | datanode1
10000 | datanode5
 9999 | datanode6
 9995 | datanode7
10000 | datanode8
(8 rows)
```

8. The preceding query result indicates that the data deviation across DNs is less than 10%. The data is evenly distributed and the appropriate distribution keys have been selected.

9. Delete the imported small-batch data.
    ```
    TRUNCATE TABLE staffs;
    ```

10. Import the full data to complete data migration.

4.6.3 Analyzing a Table

The execution plan generator needs to use table statistics to generate the most effective query execution plan to improve query performance. After data is imported, you are advised to run the **ANALYZE** statement to update table statistics. The statistics are stored in the **PG_STATISTIC** system catalog.

Analyzing a Table

ANALYZE supports row-store, column-store, HDFS, and OBS tables in ORC or CARBONDATA format. **ANALYZE** can also collect statistics about specified columns of a local table. For details about **ANALYZE**, see ANALYZE | ANALYSE.

Step 1 Update table statistics.

Do **ANALYZE** to the **product_info** table.
```
ANALYZE product_info;
```

----End

Automatically Analyzing a Table

GaussDB(DWS) provides automatic table analysis for the following two scenarios.

●   If **ANALYZE** is triggered because a query contains a table that has no statistics or a table whose amount of data modification reaches the threshold, and the execution plan does not use Fast Query Shipping (FQS), the GUC parameter **autoanalyze** is used to control the automatic collection of table statistics. In this case, a better execution plan is generated based on the collected statistics.

●   If **autovacuum** is set to **on**, the system periodically starts the autovacuum thread and automatically collects statistics on the tables whose amount of data modification reaches the threshold for triggering **ANALYZE** in the background.

Table 4-24 Automatically Analyzing a Table

Trigger Mode	Trigger Condition	Scaling Frequency	Control Parameter	Remarks
Synchronization	Statistics are completely missing.	At each query execution	autoanalyze	Statistics are cleared when the primary table is truncated.
Synchronization	The amount of modified data reaches the **ANALYZE** threshold.	At each query execution	autoanalyze	ANALYZE is triggered before the optimal plan is determined.
Asynchronization	The amount of modified data reaches the **ANALYZE** threshold.	Autovacuum thread polling check	autovacuum_mode, autovacuum_naptime	The lock times out in 2 seconds, and the execution times out in 5 minutes.

NOTICE

- The autoanalyze function supports only the default sampling mode and not the percentage sampling mode.

- The autoanalyze function does not collect multi-column statistics, which only supports percentage sampling.

- **AUTOANALYZE** is triggered because a query contains a table that has no statistics or a table whose amount of data modification reaches the threshold. In this case, **AUTOANALYZE** cannot be triggered for foreign tables or temporary tables with the **ON COMMIT [DELETE ROWS | DROP]** option.

- 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**.

- The autoanalyze function triggered by a scheduled **autovacuum** thread supports only row-store and column-store tables. It does not support foreign tables, HDFS tables, OBS foreign tables, temporary tables, unlogged tables, or toast tables.

- When **ANALYZE** is triggered during a query, a level-4 lock is added to all partitions in the partitioned table. The lock is released only after the transaction containing the query is committed. The level-4 lock does not block adding, deletion, modification, and query operations, but blocks partition modification operations such as **TRUNCATE**. You can set **object_mtime_record_mode** to **disable_partition** to release the partition locks in advance.

- The autovacuum function also depends on the following two GUC parameters in addition to **autovacuum**:

  - **track_counts**: This parameter must be set to **on** to enable statistics collection about the database.

  - autovacuum_max_workers: This parameter must be set to a value greater than 0 to specify the maximum number of concurrent autovacuum threads.

5 Hot and Cold Data Management

Introduction to Hot and Cold Data

In massive big data scenarios, as services and data volume increase, data storage and consumption increase. The need for data may vary in different time periods, therefore, data is managed in a hierarchical manner, improving data analysis performance and reducing service costs.

For example, in a network traffic analysis system, users may be interested in security events and network access in the last month, but seldom pay attention to data generated several months ago. In such scenarios, data can be classified into hot data and cold data based on time periods.

Hot and cold data is classified based on the data access frequency and update frequency.

- Hot data: Data that is frequently accessed and updated, has a high probability of being invoked in the future, and has high requirements on access response time.
- Cold: Data that cannot be updated or is seldom updated, seldom accessed, and has low requirements on response time.

You can define cold and hot management tables to switch cold data that meets the specified rules to OBS for storage. Cold and hot data can be automatically determined and migrated by partition.
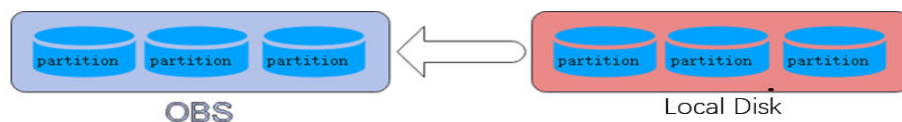
Hot and Cold Data Migration

When data is inserted to GaussDB(DWS) column-store tables, the data is first stored in hot partitions. As data accumulates, you can manually or automatically migrate the cold data to OBS for storage. The metadata, description tables, and indexes of the migrated cold data are stored locally to ensure the read performance.

Cold/Hot Switchover Policies

Currently, the hot and cold partitions can be switched based on LMT (Last Modify Time) and HPN (Hot Partition Number) policies. LMT indicates that the switchover is performed based on the last update time of the partition, and HPN indicates that the switchover is performed based on the number of reserved hot partitions.

- **LMT**: Switch the hot partition data that is not updated in the last *[day]* days to the OBS tablespace as cold partition data. *[day]* is an integer ranging from 0 to 36500, in days.

  In the following figure, *day* is set to **2**, indicating that the partitions modified in the last two days are retained as the hot partitions, while the rest is retained as the cold partitions. Assume that the current time is April 30. The delete operation is performed on the partition **[4-26]** on April 30, and the insert operation is performed on the partition **[4-27]** on April 29. Therefore, partitions **[4-26][4-27][4-29][4-30]** are retained as hot partitions.

- HPN: indicates the number of hot partitions to be reserved. The partitions are sequenced based on partition sequence IDs. The sequence ID of a partition is a built-in sequence number generated based on the partition boundary values and is not shown. For a range partition, a larger boundary value indicates a larger sequence ID. For a list partition, a larger maximum enumerated value of the partition boundary indicates a larger sequence ID. During the cold and hot switchover, data needs to be migrated to OBS. HPN is an integer ranging from 0 to 1600. If HPN is set to **0**, hot partitions are not reserved. During a cold/hot switchover, all partitions with data are converted to cold partitions and stored on OBS.

  In the following figure, HPN is set to 3, indicating that the last three partitions with data are retained as the hot partitions with the rest as the cold partitions during hot and cold partition switchover.
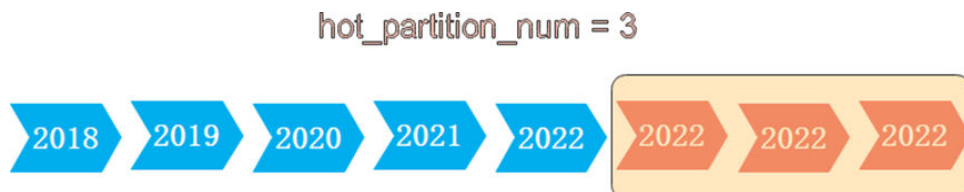
Precautions

1. Hot and cold data management supports the following functions:
   - Supports DML operations on cold and hot tables, such as **INSERT**, **COPY**, **DELETE**, **UPDATE**, and **SELECT**.
   - Supports DCL operations such as permission management on cold and hot tables.
   - Supports ANALYZE, VACUUM, MERGE INTO, and PARTITION operations on cold and hot tables.

–　Supports common column-store partitioned tables to be upgraded to hot and cold data tables.

–　Supports upgrade, scale-out, scale-in, and redistribution operations on tables with cold and hot data management enabled.

2.　Restrictions on cold and hot data management:

–　Currently, cold and hot tables support only column-store partitioned tables of version 2.0. Foreign tables do not support cold and hot partitions.

–　Only hot data can be switched to cold data. Cold data cannot be switched to hot data. If you insert data into a cold partition again, the data is directly stored in OBS. It does not turn the cold table into a hot table.

–　A partition on a DN is either hot or cold. For a partition across DNs, its data on some DNs may be hot, and some may be cold.

–　If a table has both cold and hot partitions, the query becomes slow because cold data is stored on OBS and the read/write speed are lower than those of local queries.

–　Only the cold and hot switchover policies can be modified. The tablespace of cold data in cold and hot tables cannot be modified.

–　Restrictions on partitioning cold and hot tables:

▪　Data in cold partitions cannot be exchanged.

▪　**MERGE PARTITION** supports only the merge of hot-hot partitions and cold-cold partitions.

▪　Partition operations, such as **ADD**, **MERGE**, and **SPLIT**, cannot be performed on an OBS tablespace.

▪　Tablespaces of cold and hot table partitions cannot be specified or modified during table creation.

–　Cold and hot data switchover is not performed immediately upon conditions are met. Data switchover is performed only after users manually, or through a scheduler, invoke the switchover command. Currently, the automatic scheduling time is 00:00 every day and can be modified.

–　Currently, only the LMT and HPN switchover rules are supported.

–　Cold and hot data tables do not support physical fine-grained backup and restoration. Only hot data is backed up during physical backup. Cold data on OBS does not change. The backup and restoration does not support file deletion statements, such as **TRUNCATE TABLE** and **DROP TABLE**.

Examples

1.　Create column-store cold and hot tables and set the hot data validity period LMT to 100 days.

```
CREATE TABLE lifecycle_table(i int, val text) WITH (ORIENTATION = COLUMN, storage_policy = 'LMT:
100')
PARTITION BY RANGE (i)
(
PARTITION P1 VALUES LESS THAN(5),
```

```
PARTITION P2 VALUES LESS THAN(10),
PARTITION P3 VALUES LESS THAN(15),
PARTITION P8 VALUES LESS THAN(MAXVALUE)
)ENABLE ROW MOVEMENT;
```

2. Switch cold data to the OBS tablespace.

   – Automatic switchover: The scheduler automatically triggers the switchover at 00:00 every day.

     The automatic switchover time can be customized. For example, the time can be changed to 06:30 every morning.

     ```
     select * from pg_obs_cold_refresh_time('lifecycle_table', '06:30:00');
     ```

   – Manual

     Perform the following operations to manually switch a single table:

     ```
     alter table lifecycle_table refresh storage;
     ```

     Perform the following operations to switch over all cold and hot tables in batches:

     ```
     select pg_catalog.pg_refresh_storage();
     ```

3. View data distribution in hot and cold tables.

   View the data distribution in a single table:

   ```
   select * from pg_catalog.pg_lifecycle_table_data_distribute('lifecycle_table');
   ```

   View data distribution in all hot and cold tables.

   ```
   select * from pg_catalog.pg_lifecycle_node_data_distribute();
   ```

6 Syntax Compatibility Differences Among Oracle, Teradata, and MySQL

GaussDB(DWS) is compatible with Oracle, Teradata and MySQL syntax, of which the syntax behavior is different.

Table 6-1 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
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**.

Compatibility Item	Oracle	Teradata	MySQL
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.
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.

Compatibility Item	Oracle	Teradata	MySQL
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*.
log(x)	Returns the logarithm with 10 as the base.	Returns the logarithm with 10 as the base.	Returns the natural logarithm.

Compatibility Item	Oracle	Teradata	MySQL
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.)	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.
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 time 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.

7 Database Security Management

7.1 Managing Users and Their Permissions

7.1.1 Database Users

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 Permissions**.

- 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 7-1 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}*';

7.1.2 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:

- Change the password of the user **u1**:
  ALTER USER *u1* IDENTIFIED BY '*newpassword*' REPLACE '*oldpassword*';
- Grant the **CREATEROLE** permission to user **u1**:
  ALTER USER u1 CREATEROLE;

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;
  ```

7.1.3 User-defined Password Policy

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 7-2 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 (**password_min_special**)	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. This mode is not recommended for users. ● **1** indicates that passwords are encrypted using SHA-256, which is compatible with the MD5 user authentication method of the PostgreSQL client. ● **2** indicates that passwords are encrypted using SHA-256.	2
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.	1

Password Policy	Parameter	Description	Value Range	Default Value in GaussDB(DWS)
	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 7-3 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 7-1 password_policy

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 7-2 password_effect_time

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

7.1.4 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 7-3 Hierarchical Permission Management
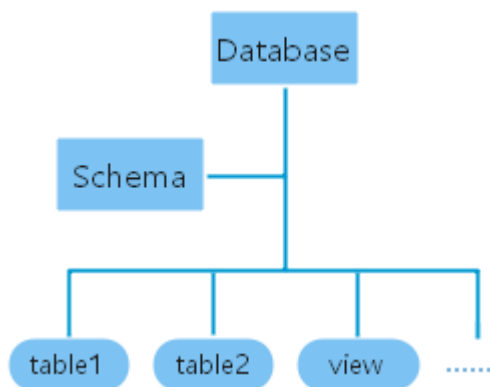
Roles

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 details about how to create, modify, and delete a role, see **CREARE 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 7-4 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.

Role	Permission
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
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 Permissions** 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;
  ```

7.1.5 Separation of Permissions

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 details about how to configure separation of permissions for, see **Separating Rights of Roles**.

For details about permission changes before and after enabling the separation of permissions, see **Table 7-5** and **Table 7-6**.

Table 7-5 Default user permissions

Object	System Administrator	Security Administrator	Audit Administrator	Common User
Tables pace	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.		
Schem a	Has permissions for all schemas.	Has all permissions for its own schemas, but does not have permissions for other users' schemas.		
Functio n	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.		
Custo mized view	Has permissions for all views.	Has permissions for its own views, but does not have permissions for other users' views.		
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 **System Catalogs and System Views**.		

Table 7-6 Changes in permissions after the separation of permissions

Objec t	System Administrator	Securi ty Admi nistra tor	Audit Admi nistra tor	Common User
Tables pace	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		

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

7.2 Sensitive Data Management

7.2.1 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

7.2.2 Data Redaction

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.

- You can create a redaction policy for a specified table and set the effective scope of the policy.

    For details, see section **CREATE REDACTION POLICY**.

- You can modify redaction policies using the provided syntax, including modifying expressions for the policy to take effect, renaming the policy, and adding, modifying, and deleting columns for data redaction.

    For details, see section **ALTER REDACTION POLICY**.

- You can delete a redaction policy by deleting data redaction function information of the policy from all columns of a table.

    For details, see section **DROP REDACTION POLICY**.

- You can use the built-in masking functions **MASK_NONE**, **MASK_FULL**, and **MASK_PARTIAL**, or create your own masking functions by using the C language or PL/pgSQL.

    For details, see section **Data Redaction Functions**.

- 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**, administrator **alice**, and common users **matu** and **july** as examples to describe the data redaction process. The user

alice is the owner of the **emp** table. The **emp** table contains private data such as the employee name, mobile number, email address, bank card number, and salary.

1. Create users **alice**, **matu**, and **july**:
   ```
   CREATE ROLE alice PASSWORD 'password';
   CREATE ROLE matu PASSWORD 'password';
   CREATE ROLE july PASSWORD 'password';
   ```

2. Create the **emp** table as user **alice**, and insert three employee records into the table.
   ```
   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');
   ```

3. User **alice** grants the **emp** table read permission to users **matu** and **july**.
   ```
   GRANT SELECT ON emp TO matu, july;
   ```

4. Only user **alice** can view all employee information. Users **matu** and **july** cannot view bank card numbers and salary data of the employees. Create a redaction policy for the **emp** table and bind the redaction function to **card_no**, **card_string**, and **salary** columns, respectively.
   ```
   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);
   ```

5. Switch to users **matu** and **july** and view the **emp** table, respectively.
   ```
   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)
   ```

6. User **matu** also has the permission to view all employee information. Only user **july** is not allowed to view the information. Modify the effective scope of the redaction policy.
   ```
   ALTER REDACTION POLICY mask_emp ON emp WHEN(current_user = 'july');
   ```

7. 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)
```

8. The information in the **phone_no**, **email**, and **birthday** columns is private data. Update redaction policy **mask_emp** and add three redaction columns.
   ```
   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);
   ```

9. Switch to user **july** and view **emp** table data.
   ```
   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)
   ```

10. 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)
    ```

11. 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 **User-Defined Functions**.

```
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)
```

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

```
DROP REDACTION POLICY mask_emp ON emp;
```

7.2.3 Using Functions for Encryption and Decryption

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)
```

8 Development and Design Proposal

8.1 Development and Design Proposal

This chapter describes the design specifications for database modeling and application development. Modeling compliant with these specifications fits the distributed processing architecture of GaussDB(DWS) and provides efficient SQL code.

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

- **Proposal**: Design rules. Services compliant with the rules can run efficiently, and those violating the rules may have low performance or logic errors.

- **Notice**: Details requiring attention during service development. This term identifies SQL behavior that complies with SQL standards but users may have misconceptions about, and default behavior that users may be unaware of in a program.

8.2 Database Object Naming Conventions

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

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

  ◯ **NOTE**

  To query the keywords of GaussDB(DWS), run **select * from pg_get_keywords()** or refer to section "Keyword."

- [Proposal] Do not use a string enclosed in double quotation marks ("") to define the database object name, unless you need to specify its capitalization. Case sensitivity of database object names makes problem location difficult.

- [Proposal] Use the same naming format for database objects.

  – In a system undergoing incremental development or service migration, you are advised to comply with its historical naming conventions.

- 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*.

8.3 Database Object Design

8.3.1 Database and Schema Design

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.

8.3.2 Table Design

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 customer's service type. For details, see **Table 8-1**.

Table 8-1 Table storage modes and scenarios

Storage Mode	Application Scenarios
Row storage	• Point queries (simple index-based queries that only return a few records) • Scenarios requiring frequent addition, deletion, and modification

Storage Mode	Application Scenarios
Column storage	● Statistical analysis queries (requiring a large number of association and grouping operations) ● Ad hoc queries (using uncertain query conditions and unable to utilize indexes to scan row-store tables)

Selecting a Distribution Mode

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

Table 8-2 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.

8.3.3 Column Design

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 8-3 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

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

8.3.4 Constraint Design

DEFAULT and NULL Constraints

- [Proposal] If all the column values can be obtained from services, you are not advised to use the **DEFAULT** constraint, 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*.

8.3.5 View and Joined Table Design

View Design

- [Proposal] Do not nest views unless they have strong dependency on each other.

- [Proposal] Try to avoid 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 joined columns should indicate their relationship. For example, they can use the same name.

8.4 JDBC Configuration

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 details about how to use CopyManager, see **CopyManager**.

8.5 SQL Compilation

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 delete objects using **DROP...CASCADE**, unless the dependency between objects is specified. Otherwise, the 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] Filtering conditions in **WHERE** clauses should comply with unilateral rules. That is, when the column name is placed on one side of a comparison operator, the optimizer automatically performs 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. 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 tablename
  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 (va11), (val2), (val3…)** statement. The optimizer will automatically convert the **IN** constraint into a non-correlated subquery to improve the query performance.

- [Proposal] Replace **(not) in** with **(not) exist** when associated columns do not contain **NULL** values. For example, in the following query statement, if the T1.C1 column does not contain any NULL value, add the NOT NULL constraint to the T1.C1 column, and then rewrite the statements.

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

Rewrite the statement as follows:

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

📖 **NOTE**

- If the value of the T1.C1 column will possibly be NULL, the preceding rewriting cannot be performed.
- 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.

8.6 User-defined External Function Usage (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 8-4 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

GaussDB(DWS)	Java
TIME	java.sql.Time (stored value treated as local time)
TIMETZ	java.sql.Time
TIMESTAMP	java.sql.Timestamp
TIMESTAMPTZ	java.sql.Timestamp

8.7 PL/pgSQL Usage

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.   A cursor variable shall be closed after being used. If the cursor variable needs to transfer data to an invoked application, the cursor shall be closed in the application. If the cursor variable is used only in a stored procedure, the cursor shall be closed 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:

        a.   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. 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 input and output parameter format of a procedure or function is **IO**_*Parameter_name*. The parameter name shall be in uppercase.

    d. Variables used in procedures and functions shall be composed of **v**_*Variable_name*. The variable name shall be in lower case.

    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.

9 Guide: JDBC- or ODBC-Based Development

9.1 Development Specifications

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

- If GUC parameters are set in the connection, 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.

9.2 Downloading Drivers

For details, see **Downloading the JDBC or ODBC Driver**.

9.3 JDBC-Based Development

Java Database Connectivity (JDBC) is a Java API for executing SQL statements, providing a unified access interface for different relational databases, based on which applications process data. GaussDB(DWS) supports JDBC 4.0 and requires JDK 1.6 or later for code compiling. It does not support JDBC-ODBC Bridge.

9.3.1 JDBC Package and Driver Class

JDBC Package

Obtain the package **dws_8.1.x_jdbc_driver.zip** from the management console. For details, see **Downloading Drivers**.

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.

9.3.2 Development Process

Figure 9-1 JDBC-based application development process
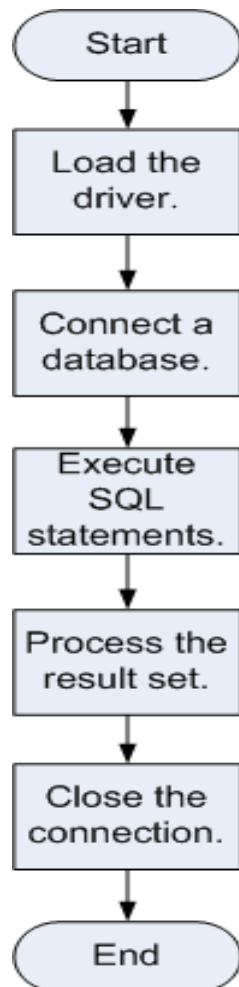
9.3.3 Loading a Driver

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"**.

9.3.4 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 **Password Reset**.

   - 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);

Parameter

Table 9-1 Database connection parameters

Parameter	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 the host is on a different network segment from the GaussDB(DWS) it connects to, set the IP address of the server specified by **host** to the value of **mppdb.coo.cooListenIp2** (IP address accessed by applications) set on FusionInsight Manager. 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 JDBC program on the host where the CN is located and set **host** to **127.0.0.1**. If you do not do so, the error message "FATAL: Forbid remote connection with trust method!" may be displayed. It is recommended that the service system be deployed outside the cluster. If it is deployed inside, the database performance may be affected. • **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**

Parameter	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.

Parameter	Description
password	Indicates the password of a database user.

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;
  }
```

9.3.5 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

9.3.6 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 9-2** 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 9-2 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.
resultSetConcurrency	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 9-3** lists these methods.

Table 9-3 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 9-4** lists the method.

Table 9-4 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 9-5** lists the common methods for obtaining data. If you want to know more about other methods, see JDK official documents.

Table 9-5 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.

9.3.7 Closing the Connection

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

Call the close method to close the connection, such as, **conn. close()**.

9.3.8 Example: Common Operations

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.

```java
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();
```

9.3.9 Example: Retrying SQL Queries for Applications

If the primary DN is faulty and cannot be restored within 40s, its standby is automatically promoted to primary to ensure the normal running of the cluster. Jobs running during the failover will fail and those started after the failover 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;
   }
```

```java
//Run an ordinary SQL statement. Create a 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();
    }
    //Perform 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);

            //Disconnect from the database.
            try {
                conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
```

```
    }

  }
```

9.3.10 Example: 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);
       }

     //Export the query result of SELECT * FROM migration_table to the local file d:/data.txt.
     try {
     copyToFile(conn, "d:/data.txt", "(SELECT * FROM migration_table)");
 } catch (SQLException e) {
// TODO Auto-generated catch block
e.printStackTrace();
 } catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
 }
     //Import data from the d:/data.txt file to the migration_table_1 table.
     try {
     copyFromFile(conn, "d:/data.txt", tablename1);
 } catch (SQLException e) {
// TODO Auto-generated catch block
         e.printStackTrace();
 } catch (IOException e) {
 // TODO Auto-generated catch block
```

```
       e.printStackTrace();
}

       //Export the data from the migration_table_1 table to the d:/data1.txt file.
       try {
       copyToFile(conn, "d:/data1.txt", tablename1);
 } catch (SQLException e) {
// TODO Auto-generated catch block
e.printStackTrace();
 } catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
    }

   public static void copyFromFile(Connection connection, String filePath, String tableName)
        throws SQLException, IOException {

      FileInputStream fileInputStream = null;

      try {
         CopyManager copyManager = new CopyManager((BaseConnection)connection);
         fileInputStream = new FileInputStream(filePath);
         copyManager.copyIn("COPY " + tableName + " FROM STDIN", fileInputStream);
      } finally {
         if (fileInputStream != null) {
            try {
               fileInputStream.close();
            } catch (IOException e) {
               e.printStackTrace();
            }
         }
      }
   }

   public static void copyToFile(Connection connection, String filePath, String tableOrQuery)
        throws SQLException, IOException {

      FileOutputStream fileOutputStream = null;

      try {
         CopyManager copyManager = new CopyManager((BaseConnection)connection);
         fileOutputStream = new FileOutputStream(filePath);
         copyManager.copyOut("COPY " + tableOrQuery + " TO STDOUT", fileOutputStream);
      } finally {
         if (fileOutputStream != null) {
            try {
               fileOutputStream.close();
            } catch (IOException e) {
               e.printStackTrace();
            }
         }
      }
   }
}
```

9.3.11 Example: 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;
```

```java
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");          //DWS username
        String pass = new String(" ********");     //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;
    }
}
```

9.3.12 JDBC Interface Reference

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 JDBC4.0.

9.3.12.1 java.sql.Connection

This section describes **java.sql.Connection**, the interface for connecting to a database.

Table 9-6 Support status for java.sql.Connection

Method Name	Return Type	Support JDBC 4
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

Method Name	Return Type	Support JDBC 4
setClientInfo(String name,String value)	void	Yes

NOTICE

The AutoCommit mode is used by default within the interface. If you disable it running **setAutoCommit(false)**, all the statements executed later will be packaged in explicit transactions, and you cannot execute statements that cannot be executed within transactions.

9.3.12.2 java.sql.CallableStatement

This section describes **java.sql.CallableStatement**, the stored procedure execution interface.

Table 9-7 Support status for java.sql.CallableStatement

Method Name	Return Type	Support JDBC 4
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

Method Name	Return Type	Support JDBC 4
getTimestamp(int parameterIndex)	Timestamp	Yes
getObject(int parameterIndex)	Object	Yes

☐ **NOTE**

- The batch operation of statements containing OUT parameter is not allowed.
- The following methods are inherited from java.sql.Statement: close, execute, executeQuery, executeUpdate, getConnection, getResultSet, getUpdateCount, isClosed, setMaxRows, and setFetchSize.
- The following methods are inherited from java.sql.PreparedStatement: addBatch, clearParameters, execute, executeQuery, executeUpdate, getMetaData, setBigDecimal, setBoolean, setByte, setBytes, setDate, setDouble, setFloat, setInt, setLong, setNull, setObject, setString, setTime, and setTimestamp.

9.3.12.3 java.sql.DatabaseMetaData

This section describes **java.sql.DatabaseMetaData**, the interface for defining database objects.

Table 9-8 Support status for java.sql.DatabaseMetaData

Method Name	Return Type	Support JDBC 4
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

Method Name	Return Type	Support JDBC 4
getDatabaseProductVersion()	String	Yes
getDriverName()	String	Yes
getDriverVersion()	String	Yes
getDriverMajorVersion()	int	Yes
getDriverMinorVersion()	int	Yes
usesLocalFiles()	boolean	Yes
usesLocalFilePerTable()	boolean	Yes
supportsMixedCaseIdentifiers()	boolean	Yes
storesUpperCaseIdentifiers()	boolean	Yes
storesLowerCaseIdentifiers()	boolean	Yes
supportsMixedCaseQuotedIdentifiers()	boolean	Yes
storesUpperCaseQuotedIdentifiers()	boolean	Yes
storesLowerCaseQuotedIdentifiers()	boolean	Yes
storesMixedCaseQuotedIdentifiers()	boolean	Yes
supportsAlterTableWithAddColumn()	boolean	Yes
supportsAlterTableWithDropColumn()	boolean	Yes
supportsColumnAliasing()	boolean	Yes
nullPlusNonNullIsNull()	boolean	Yes
supportsConvert()	boolean	Yes
supportsConvert(int fromType, int toType)	boolean	Yes
supportsTableCorrelationNames()	boolean	Yes
supportsDifferentTableCorrelationNames()	boolean	Yes
supportsExpressionsInOrderBy()	boolean	Yes

Method Name	Return Type	Support JDBC 4
supportsOrderByUnrelated()	boolean	Yes
supportsGroupBy()	boolean	Yes
supportsGroupByUnrelated()	boolean	Yes
supportsGroupByBeyondSelect()	boolean	Yes
supportsLikeEscapeClause()	boolean	Yes
supportsMultipleResultSets()	boolean	Yes
supportsMultipleTransactions()	boolean	Yes
supportsNonNullableColumns()	boolean	Yes
supportsMinimumSQLGrammar()	boolean	Yes
supportsCoreSQLGrammar()	boolean	Yes
supportsExtendedSQLGrammar()	boolean	Yes
supportsANSI92EntryLevelSQL()	boolean	Yes
supportsANSI92IntermediateSQL()	boolean	Yes
supportsANSI92FullSQL()	boolean	Yes
supportsIntegrityEnhancementFacility()	boolean	Yes
supportsOuterJoins()	boolean	Yes
supportsFullOuterJoins()	boolean	Yes
supportsLimitedOuterJoins()	boolean	Yes
isCatalogAtStart()	boolean	Yes
supportsSchemasInDataManipulation()	boolean	Yes
supportsSavepoints()	boolean	Yes
supportsResultSetHoldability(int holdability)	boolean	Yes
getResultSetHoldability()	int	Yes
getDatabaseMajorVersion()	int	Yes

Method Name	Return Type	Support JDBC 4
getDatabaseMinorVersion()	int	Yes
getJDBCMajorVersion()	int	Yes
getJDBCMinorVersion()	int	Yes

9.3.12.4 java.sql.Driver

This section describes **java.sql.Driver**, the database driver interface.

Table 9-9 Support status for java.sql.Driver

Method Name	Return Type	Support JDBC 4
acceptsURL(String url)	boolean	Yes
connect(String url, Properties info)	Connection	Yes
jdbcCompliant()	boolean	Yes
getMajorVersion()	int	Yes
getMinorVersion()	int	Yes

9.3.12.5 java.sql.PreparedStatement

This section describes **java.sql.PreparedStatement**, the interface for preparing statements.

Table 9-10 Support status for java.sql.PreparedStatement

Method Name	Return Type	Support JDBC 4
clearParameters()	void	Yes
execute()	boolean	Yes
executeQuery()	ResultSet	Yes
excuteUpdate()	int	Yes
getMetaData()	ResultSetMetaData	Yes
setBoolean(int parameterIndex, boolean x)	void	Yes
setBigDecimal(int parameterIndex, BigDecimal x)	void	Yes

Method Name	Return Type	Support JDBC 4
setByte(int parameterIndex, byte x)	void	Yes
setBytes(int parameterIndex, byte[] x)	void	Yes
setDate(int parameterIndex, Date x)	void	Yes
setDouble(int parameterIndex, double x)	void	Yes
setFloat(int parameterIndex, float x)	void	Yes
setInt(int parameterIndex, int x)	void	Yes
setLong(int parameterIndex, long x)	void	Yes
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

- Execute addBatch() and execute() only after running clearBatch().
- Batch is not cleared by calling executeBatch(). Clear batch by explicitly calling clearBatch().
- After bounded variables of a batch are added, if you want to reuse these values (add a batch again), set*() is not necessary.
- The following methods are inherited from java.sql.Statement: close, execute, executeQuery, executeUpdate, getConnection, getResultSet, getUpdateCount, isClosed, setMaxRows, and setFetchSize.

9.3.12.6 java.sql.ResultSet

This section describes **java.sql.ResultSet**, the interface for execution result sets.

Table 9-11 Support status for java.sql.ResultSet

Method Name	Return Type	Support JDBC 4
findColumn(String columnLabel)	int	Yes
getBigDecimal(int columnIndex)	BigDecimal	Yes
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

Method Name	Return Type	Support JDBC 4
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
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

- One Statement cannot have multiple open ResultSets.
- The cursor that is used for traversing the ResultSet cannot be open after committed.

9.3.12.7 java.sql.ResultSetMetaData

This section describes **java.sql.ResultSetMetaData**, which provides details about ResultSet object information.

Table 9-12 Support status for java.sql.ResultSetMetaData

Method Name	Return Type	Support JDBC 4
getColumnCount()	int	Yes
getColumnName(int column)	String	Yes

Method Name	Return Type	Support JDBC 4
getColumnType(int column)	int	Yes
getColumnTypeName(int column)	String	Yes

9.3.12.8 java.sql.Statement

This section describes **java.sql.Statement**, the interface for executing SQL statements.

Table 9-13 Support status for java.sql.Statement

Method Name	Return Type	Support JDBC 4
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

Using setFetchSize can reduce the memory occupied by result sets on the client. Result sets are packaged into cursors and segmented for processing, which will increase the communication traffic between the database and the client, affecting performance.

Database cursors are valid only within their transaction. If **setFetchSize** is set, set **setAutoCommit(false)** and commit transactions on the connection to flush service data to a database.

9.3.12.9 javax.sql.ConnectionPoolDataSource

This section describes **javax.sql.ConnectionPoolDataSource**, the interface for data source connection pools.

Table 9-14 Support status for javax.sql.ConnectionPoolDataSource

Method Name	Return Type	Support JDBC 4
getLoginTimeout()	int	Yes
getLogWriter()	PrintWriter	Yes
getPooledConnection()	PooledConnection	Yes
getPooledConnec-tion(String user,String password)	PooledConnection	Yes
setLoginTimeout(int seconds)	void	Yes
setLogWriter(PrintWriter out)	void	Yes

9.3.12.10 javax.sql.DataSource

This section describes **javax.sql.DataSource**, the interface for data sources.

Table 9-15 Support status for javax.sql.DataSource

Method Name	Return Type	Support JDBC 4
getConneciton()	Connection	Yes
getConnection(String username,String password)	Connection	Yes
getLoginTimeout()	int	Yes
getLogWriter()	PrintWriter	Yes
setLoginTimeout(int seconds)	void	Yes
setLogWriter(PrintWriter out)	void	Yes

9.3.12.11 javax.sql.PooledConnection

This section describes **javax.sql.PooledConnection**, the connection interface created by a connection pool.

Table 9-16 Support status for javax.sql.PooledConnection

Method Name	Return Type	Support JDBC 4
addConnectionEventListener (ConnectionEventListener listener)	void	Yes
close()	void	Yes
getConnection()	Connection	Yes
removeConnectionEventListener (ConnectionEventListener listener)	void	Yes
addStatementEventListener (StatementEventListener listener)	void	Yes
removeStatementEventListener (StatementEventListener listener)	void	Yes

9.3.12.12 javax.naming.Context

This section describes **javax.naming.Context**, the context interface for connection configuration.

Table 9-17 Support status for javax.naming.Context

Method Name	Return Type	Support JDBC 4
bind(Name name, Object obj)	void	Yes
bind(String name, Object obj)	void	Yes
lookup(Name name)	Object	Yes
lookup(String name)	Object	Yes
rebind(Name name, Object obj)	void	Yes
rebind(String name, Object obj)	void	Yes
rename(Name oldName, Name newName)	void	Yes
rename(String oldName, String newName)	void	Yes
unbind(Name name)	void	Yes
unbind(String name)	void	Yes

9.3.12.13 javax.naming.spi.InitialContextFactory

This section describes **javax.naming.spi.InitialContextFactory**, the initial context factory interface.

Table 9-18 Support status for javax.naming.spi.InitialContextFactory

Method Name	Return Type	Support JDBC 4
getInitialContext(Hashtable<?,?> environment)	Context	Yes

9.3.12.14 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

Basic Methods

Table 9-19 Common methods of CopyManager

Return Value	Method	Description	throws
CopyIn	copyIn(String sql)	-	SQLException
long	copyIn(String sql, InputStream from)	Uses **COPY FROM STDIN** to quickly load data to tables in the database from InputStream.	SQLException,IOException

Return Value	Method	Description	throws
long	copyIn(String sql, InputStream from, int bufferSize)	Uses **COPY FROM STDIN** to quickly load data to tables in the database from InputStream.	SQLException,IOEx ception
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

9.4 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 9-2 shows the system structure of ODBC.

Figure 9-2 ODBC system structure
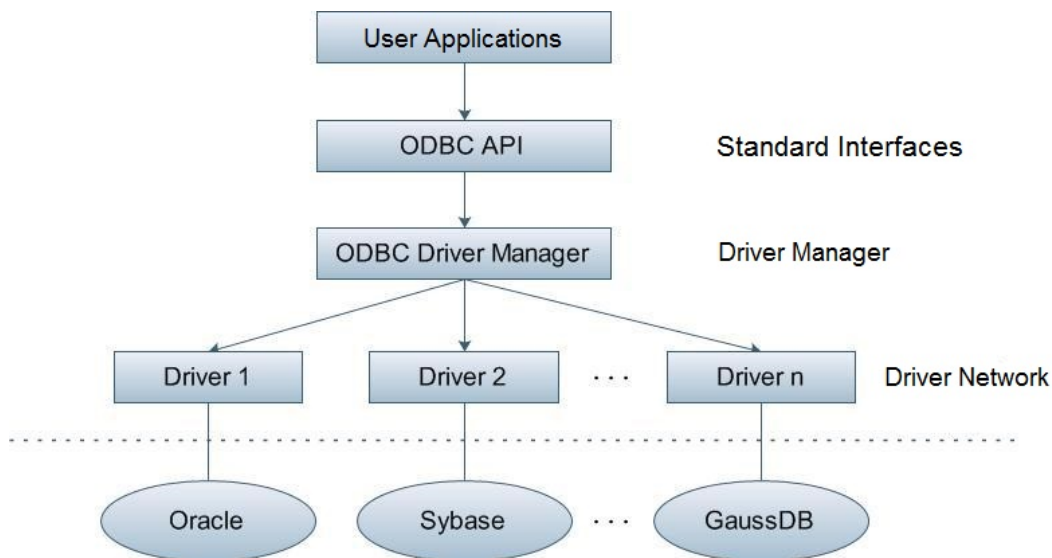
GaussDB(DWS) supports ODBC 3.5 in the following environments.

Table 9-20 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 Huawei-developed data types, such as tinyint, smalldatetime, and nvarchar2.

9.4.1 ODBC Package and Its Dependent Libraries and Header Files

ODBC Package for the Linux OS

Obtain the **dws_8.1.***x***_odbc_driver_for_***xxx_xxx*.**zip** package from the release 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 release package. In the Windows OS, the required header files and library files are system-resident.

9.4.2 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:

http://sourceforge.net/projects/unixodbc/files/unixODBC/2.3.0/unixODBC-2.3.0.tar.gz/download

Step 2 Currently, unixODBC-2.2.1 is not supported. Assume you are to install unixODBC-2.3.0. Run the following commands. unixODBC is installed in the **/usr/local** directory by default. The data source file is generated in the **/usr/local/etc** directory, and the library file is generated in the **/usr/local/lib** directory.

```
tar zxvf unixODBC-2.3.0.tar.gz
cd unixODBC-2.3.0
# Open the configure file. If it does not exist, open the configure.ac file. Find LIB_VERSION.
# Change the value of LIB_VERSION to 1:0:0 to compile a *.so.1 dynamic library with the same dependency
```

on **psqlodbcw.so**.
```
vim configure

./configure --enable-gui=no # To perform the compilation on a TaiShan server, add the configure
parameter --build=aarch64-unknown-linux-gnu.
make
# The installation may require root permissions.
make install
```

Install unixODBC. If another version of unixODBC has been installed, it will be overwritten after installation.

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 9-21**.

   Table 9-21 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 9-22**.

Table 9-22 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
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

Parameter	Description	Example
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
ForExtensionCon-nector	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
KeepDisallowPre-mature	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**.	KeepDisallowPrema-ture=1 **NOTE** – When **UseServerSidePrepare** is set to **1**, the **KeepDisallowPrema-ture** parameter does not take effect. To use this parameter, set **UseServerSidePrepare** to **0**. For example, set **UseDeclareFetch** to **1**. KeepDisallowPrema-ture=1 UseServerSidePre-pare=0

The valid values of **sslmode** are as follows.

Table 9-23 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 9-23**.

- 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.

9.4.3 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. **C:\Windows\System32** stores the environment consistent with the current OS. For technical details, see Windows technical documents.

- 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)**.

- 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.)
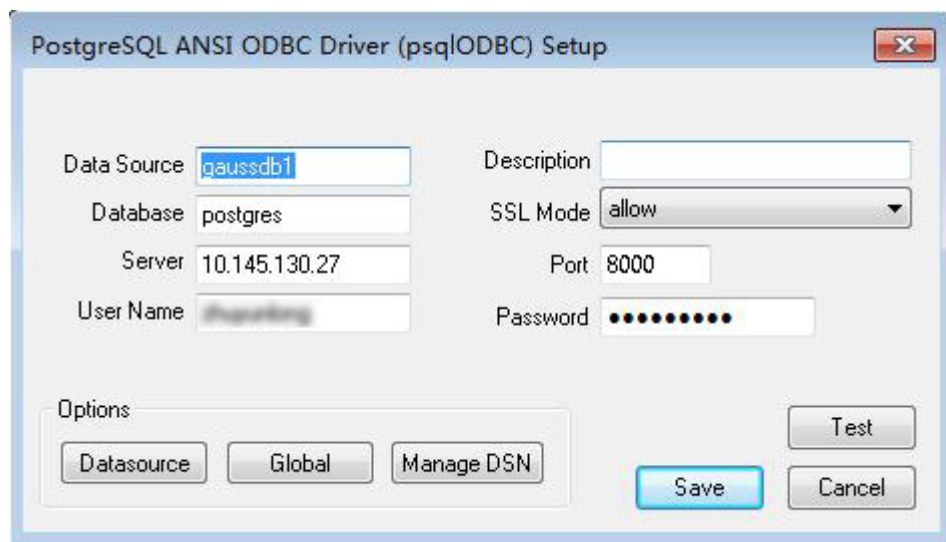
NOTICE

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 9-24 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. **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.
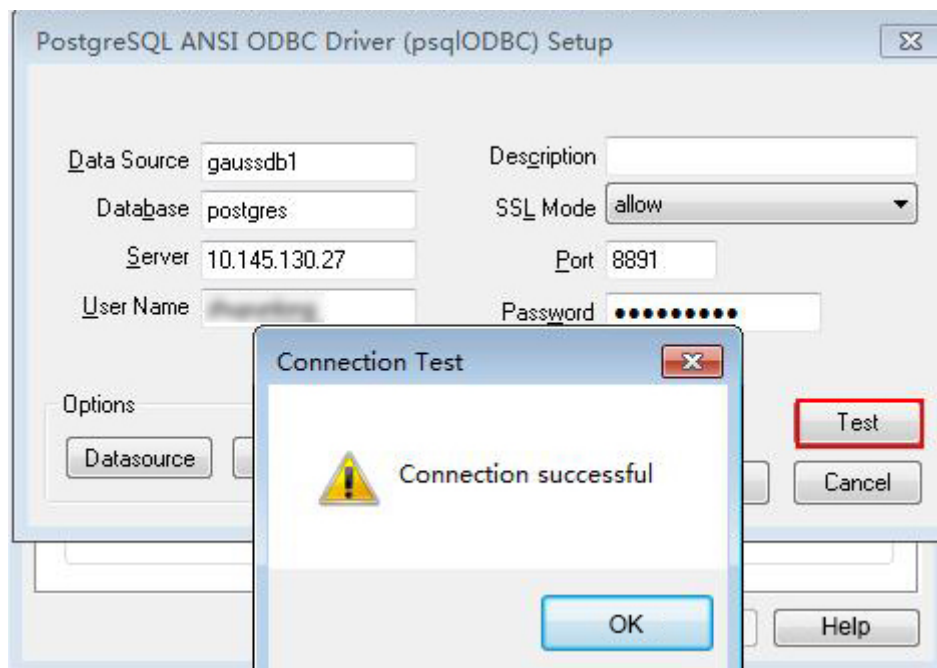
- If error information is displayed, the configuration is incorrect. Check the configuration.

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 9-24**.

- 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 **Password Reset**.

     ▪ 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.

9.4.4 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. Executes 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 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 ready 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

```
/*********************************************************************
* Set UseBatchProtocol to 1 in the data source and set the database parameter support_batch_bind
* to on.
* The CHECK_ERROR command is used to check and print error information.
* This example is used to interactively obtain the DSN, data volume to be processed, and volume of ignored
data from users, and insert required data into the test_odbc_batch_insert table.
*********************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <sql.h>
#include <sqlext.h>
#include <string.h>

#include "util.c"

void Exec(SQLHDBC hdbc, SQLCHAR* sql)
{
    SQLRETURN retcode;              // Return status
    SQLHSTMT hstmt = SQL_NULL_HSTMT;    // Statement handle
    SQLCHAR     loginfo[2048];

    // Allocate Statement Handle
    retcode = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
    CHECK_ERROR(retcode, "SQLAllocHandle(SQL_HANDLE_STMT)",
            hstmt, SQL_HANDLE_STMT);

    // Prepare Statement
    retcode = SQLPrepare(hstmt, (SQLCHAR*) sql, SQL_NTS);
    sprintf((char*)loginfo, "SQLPrepare log: %s", (char*)sql);
    CHECK_ERROR(retcode, loginfo, hstmt, SQL_HANDLE_STMT);

    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 Verion
        retcode = SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION,
                                (SQLPOINTER*)SQL_OV_ODBC3, 0);
        CHECK_ERROR(retcode, "SQLSetEnvAttr(SQL_ATTR_ODBC_VERSION)",
                henv, SQL_HANDLE_ENV);

        // Allocate Connection
        retcode = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
        CHECK_ERROR(retcode, "SQLAllocHandle(SQL_HANDLE_DBC)",
                henv, SQL_HANDLE_DBC);

        // Set Login Timeout
        retcode = SQLSetConnectAttr(hdbc, SQL_LOGIN_TIMEOUT, (SQLPOINTER)5, 0);
        CHECK_ERROR(retcode, "SQLSetConnectAttr(SQL_LOGIN_TIMEOUT)",
                hdbc, SQL_HANDLE_DBC);

        // Set Auto Commit
        retcode = SQLSetConnectAttr(hdbc, SQL_ATTR_AUTOCOMMIT,
                                (SQLPOINTER)(1), 0);
        CHECK_ERROR(retcode, "SQLSetConnectAttr(SQL_ATTR_AUTOCOMMIT)",
                hdbc, SQL_HANDLE_DBC);

        // Connect to DSN
        sprintf(loginfo, "SQLConnect(DSN:%s)", dsn);
        retcode = SQLConnect(hdbc, (SQLCHAR*) dsn, SQL_NTS,
                        (SQLCHAR*) NULL, 0, NULL, 0);
        CHECK_ERROR(retcode, loginfo, hdbc, SQL_HANDLE_DBC);

        // init table info.
        Exec(hdbc, "drop table if exists test_odbc_batch_insert");
        Exec(hdbc, "create table test_odbc_batch_insert(id int primary key, col varchar2(50))");

// The following code constructs the data to be inserted based on the data volume entered by users:
    {
        SQLRETURN retcode;
        SQLHSTMT hstmtinesrt = SQL_NULL_HSTMT;
        int          i;
        SQLCHAR      *sql = NULL;
        SQLINTEGER   *ids  = NULL;
        SQLCHAR      *cols = NULL;
        SQLLEN       *bufLenIds = NULL;
        SQLLEN       *bufLenCols = NULL;
        SQLUSMALLINT *operptr = NULL;
        SQLUSMALLINT *statusptr = NULL;
        SQLULEN      process = 0;

// Data is constructed by column. Each column is stored continuously.
        ids = (SQLINTEGER*)malloc(sizeof(ids[0]) * batchCount);
        cols = (SQLCHAR*)malloc(sizeof(cols[0]) * batchCount * 50);
// Data size in each row for a column
        bufLenIds = (SQLLEN*)malloc(sizeof(bufLenIds[0]) * batchCount);
        bufLenCols = (SQLLEN*)malloc(sizeof(bufLenCols[0]) * batchCount);
// Whether this row needs to be processed. The value is SQL_PARAM_IGNORE or SQL_PARAM_PROCEED.
        operptr = (SQLUSMALLINT*)malloc(sizeof(operptr[0]) * batchCount);
        memset(operptr, 0, sizeof(operptr[0]) * batchCount);
```

```
// Processing result of the row
// Note: In the database, a statement belongs to one transaction. Therefore, data is processed as a unit.
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;
}
```

9.4.5 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).

9.4.5.1 SQLAllocEnv

In ODBC 3.x, **SQLAllocEnv** (an ODBC 2.x function) was deprecated and replaced with **SQLAllocHandle**. For details, see **SQLAllocHandle**.

9.4.5.2 SQLAllocConnect

In ODBC 3.x, **SQLAllocConnect** (an ODBC 2.x function) was deprecated and replaced with **SQLAllocHandle**. For details, see **SQLAllocHandle**.

9.4.5.3 SQLAllocHandle

Function

SQLAllocHandle allocates environment, connection, or statement handles. This function is a generic function for allocating handles that replaces the deprecated ODBC 2.x functions **SQLAllocEnv**, **SQLAllocConnect**, and **SQLAllocStmt**.

Prototype

```
SQLRETURN SQLAllocHandle(SQLSMALLINT   HandleType,
                 SQLHANDLE    InputHandle,
                 SQLHANDLE    *OutputHandlePtr);
```

Parameter

Table 9-25 SQLAllocHandle parameters

Keyword	Description
HandleType	The type of handle to be allocated by SQLAllocHandle. The value must be one of the following: ● SQL_HANDLE_ENV (environment handle) ● SQL_HANDLE_DBC (connection handle) ● SQL_HANDLE_STMT (statement handle) ● SQL_HANDLE_DESC (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.
InputHandle	Existing handle to use as a context for the new handle being allocated. ● If **HandleType** is **SQL_HANDLE_ENV**, this 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 to return the handle to the newly allocated data structure.

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. Values returned by other APIs are similar to the preceding values.

Precautions

When allocating a non-environment handle, if **SQLAllocHandle** returns **SQL_ERROR**, it sets **OutputHandlePtr** to **SQL_NULL_HENV**, **SQL_NULL_HDBC**, **SQL_NULL_HSTMT**, or **SQL_NULL_HDESC**. The application can then call **SQLGetDiagRec**, with **HandleType** and **Handle** set to **IntputHandle**, to obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Examples

See **Examples**.

9.4.5.4 SQLAllocStmt

In ODBC 3.x, **SQLAllocStmt** was deprecated and replaced with **SQLAllocHandle**. For details, see **SQLAllocHandle**.

9.4.5.5 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);
```

Parameter

Table 9-26 SQLBindCol parameters

Keyword	Description
StatementHandle	Statement handle.
ColumnNumber	Number of the column to be bound. The column number starts with 0 and increases in ascending order. Column 0 is the bookmark column. If no bookmark column is set, column numbers start at 1.
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	Size of the **TargetValuePtr** buffer in bytes available to store the column data.
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 succeeded.
- SQL_SUCCESS_WITH_INFO indicates some warning information is displayed.
- SQL_ERROR indicates major errors, such as memory allocation and connection failures.
- SQL_INVALID_HANDLE indicates that invalid handles were called. Values returned by other APIs are similar to the preceding values.

Precautions

If **SQLBindCol** returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can then call **SQLGetDiagRec**, with **HandleType** and **Handle** set to **SQL_HANDLE_STMT** and **StatementHandle**, respectively, to obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Examples

See **Examples**.

9.4.5.6 SQLBindParameter

Function

SQLBindParameter is used to associate (bind) parameter markers in an SQL statement to a buffer.

Prototype

```
SQLRETURN SQLBindParameter(SQLHSTMT        StatementHandle,
                  SQLUSMALLINT   ParameterNumber,
                  SQLSMALLINT    InputOutputType,
                  SQLSMALLINT    ValuetType,
                  SQLSMALLINT    ParameterType,
                  SQLULEN        ColumnSize,
                  SQLSMALLINT    DecimalDigits,
                  SQLPOINTER     ParameterValuePtr,
                  SQLLEN         BufferLength,
                  SQLLEN         *StrLen_or_IndPtr);
```

Parameter

Table 9-27 SQLBindParameter

Keyword	Description
StatementHandle	Statement handle.
ParameterNumber	Parameter marker number, starting at 1 and increasing in an ascending order.
InputOutputType	Input/output type of the parameter.
ValueType	C data type of the parameter.
ParameterType	SQL data type of the parameter.
ColumnSize	Size of the column or expression of the corresponding parameter marker.
DecimalDigits	Digital number of the column or the expression of the corresponding parameter marker.
ParameterValuePtr	Pointer to the storage parameter buffer.
BufferLength	Size of the ParameterValuePtr buffer in bytes.
StrLen_or_IndPtr	Pointer to the length or indicator of the buffer. If StrLen_or_IndPtr is null, no length or indicator is used.

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates some warning information is displayed.

- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. Values returned by other APIs are similar to the preceding values.

Precautions

If **SQLBindCol** returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can then call **SQLGetDiagRec**, with **HandleType** and **Handle** set to **SQL_HANDLE_STMT** and **StatementHandle**, respectively, to obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Examples

See **Examples**.

9.4.5.7 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);
```

Parameter

Table 9-28 SQLColAttribute parameter

Keyword	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.
CharacterAttribu-tePtr	**Output parameter**: pointer to the buffer that returns FieldIdentifier field value.

Keyword	Description
BufferLength	• **FieldIdentifier** indicates the length of the buffer if **FieldIdentifier** is an ODBC-defined field and **CharacterAttributePtr** points to a character string or a binary buffer. • Ignore this parameter if **FieldIdentifier** is an ODBC-defined field and **CharacterAttributePtr** points to an integer.
StringLengthPtr	**Output parameter**: pointer to a buffer in which the total number of valid bytes (for string data) is stored in ***CharacterAttributePtr**. Ignore the value of **BufferLength** if the data is not a string.
NumericAttributePtr	**Output parameter**: pointer to an integer buffer in which the value of the **FieldIdentifier** field in the **ColumnNumber** row of the IRD is returned.

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. Values returned by other APIs are similar to the preceding values.

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. The **SQLSTATE** value provides the detailed function calling information.

Examples

See **Examples**.

9.4.5.8 SQLConnect

Function

SQLConnect establishes a connection between a driver and a data source. After the connection, the connection handle can be used to access all information about the data source, including its application operating status, transaction processing status, and error information.

Prototype

```
SQLRETURN  SQLConnect(SQLHDBC      ConnectionHandle,
               SQLCHAR       *ServerName,
```

```
SQLSMALLINT   NameLength1,
SQLCHAR       *UserName,
SQLSMALLINT   NameLength2,
SQLCHAR       *Authentication,
SQLSMALLINT   NameLength3);
```

Parameter

Table 9-29 SQLConnect parameters

Keyword	Description
ConnectionHandle	Connection handle, obtained from **SQLAllocHandle**.
ServerName	Name of the data source to connect to.
NameLength1	Length of **ServerName**.
UserName	User name of the database in the data source.
NameLength2	Length of **UserName**.
Authentication	User password of the database in the data source.
NameLength3	Length of **Authentication**.

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.

- **SQL_SUCCESS_WITH_INFO** indicates some warning information is displayed.

- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.

- **SQL_INVALID_HANDLE** indicates that invalid handles were called. Values returned by other APIs are similar to the preceding values.

- **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. The **SQLSTATE** value provides the detailed function calling information.

Examples

See **Examples**.

9.4.5.9 SQLDisconnect

Function

SQLDisconnect closes the connection associated with the database connection handle.

Prototype

```
SQLRETURN SQLDisconnect(SQLHDBC    ConnectionHandle);
```

Parameter

Table 9-30 SQLDisconnect parameters

Keyword	Description
ConnectionHandle	Connection handle, obtained from **SQLAllocHandle**.

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. Values returned by other APIs are similar to the preceding values.

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. The **SQLSTATE** value provides the detailed function calling information.

Examples

See **Examples**.

9.4.5.10 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);
```

Parameter

Table 9-31 SQLExecDirect parameters

Keyword	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 succeeded.

- **SQL_SUCCESS_WITH_INFO** indicates some warning information is displayed.

- **SQL_NEED_DATA** indicates insufficient parameters provided before executing the SQL statement.

- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.

- **SQL_INVALID_HANDLE** indicates that invalid handles were called. Values returned by other APIs are similar to the preceding values.

- **SQL_STILL_EXECUTING** indicates that the statement is being executed.

- **SQL_NO_DATA** indicates that the SQL statement does not return a result set.

Precautions

If **SQLExecDirect** returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can then call **SQLGetDiagRec**, set **HandleType** and **Handle** to **SQL_HANDLE_STMT** and **StatementHandle**, and obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Examples

See **Examples**.

9.4.5.11 SQLExecute

Function

The **SQLExecute** function executes a prepared SQL statement using **SQLPrepare**. The statement is executed using the current value of any application variables that were bound to parameter markers by **SQLBindParameter**.

Prototype

```
SQLRETURN SQLExecute(SQLHSTMT    StatementHandle);
```

Parameter

Table 9-32 SQLExecute parameters

Keyword	Description
StatementHandle	Statement handle to be executed.

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.

- **SQL_SUCCESS_WITH_INFO** indicates some warning information is displayed.

- **SQL_NEED_DATA** indicates insufficient parameters provided before executing the SQL statement.

- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.

- **SQL_NO_DATA** indicates that the SQL statement does not return a result set.

- **SQL_INVALID_HANDLE** indicates that invalid handles were called. Values returned by other APIs are similar to the preceding values.

- **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. The **SQLSTATE** value provides the detailed function calling information.

Examples

See **Examples**.

9.4.5.12 SQLFetch

Function

SQLFetch advances the cursor to the next row of the result set and retrieves any bound columns.

Prototype

```
SQLRETURN SQLFetch(SQLHSTMT    StatementHandle);
```

Parameter

Table 9-33 SQLFetch parameters

Keyword	Description
StatementHandl e	Statement handle, obtained from **SQLAllocHandle**.

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_NO_DATA** indicates that the SQL statement does not return a result set.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. Values returned by other APIs are similar to the preceding values.
- **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. The **SQLSTATE** value provides the detailed function calling information.

Examples

See **Examples**.

9.4.5.13 SQLFreeStmt

In ODBC 3.x, **SQLFreeStmt** (an ODBC 2.x function) was deprecated and replaced with **SQLFreeHandle**. For details, see **SQLFreeHandle**.

9.4.5.14 SQLFreeConnect

In ODBC 3.x, **SQLFreeConnect** (an ODBC 2.x function) was deprecated and replaced with **SQLFreeHandle**. For details, see **SQLFreeHandle**.

9.4.5.15 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);
```

Parameter

Table 9-34 SQLFreeHandle parameters

Keyword	Description
HandleType	The type of handle to be freed by SQLFreeHandle. The value must be one of the following: ● SQL_HANDLE_ENV ● SQL_HANDLE_DBC ● SQL_HANDLE_STMT ● SQL_HANDLE_DESC If **HandleType** is not one of the preceding values, **SQLFreeHandle** returns **SQL_INVALID_HANDLE**.
Handle	The name of the handle to be freed.

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. Values returned by other APIs are similar to the preceding values.

Precautions

If **SQLFreeHandle** returns **SQL_ERROR**, the handle is still valid.

Examples

See **Examples**.

9.4.5.16 SQLFreeEnv

In ODBC 3.x, **SQLFreeEnv** (an ODBC 2.x function) was deprecated and replaced with **SQLFreeHandle**. For details, see **SQLFreeHandle**.

9.4.5.17 SQLPrepare

Function

SQLPrepare prepares an SQL statement to be executed.

Prototype

```
SQLRETURN SQLPrepare(SQLHSTMT      StatementHandle,
              SQLCHAR     *StatementText,
              SQLINTEGER   TextLength);
```

Parameter

Table 9-35 SQLPrepare parameters

Keyword	Description
StatementHandle	Statement handle.
StatementText	SQL text string.
TextLength	Length of **StatementText**.

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. Values returned by other APIs are similar to the preceding values.
- **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. The **SQLSTATE** value provides the detailed function calling information.

Examples

See **Examples**.

9.4.5.18 SQLGetData

Function

SQLGetData retrieves data for a single column in the current row of the result set. It can be called for many times to retrieve data of variable lengths.

Prototype

```
SQLRETURN SQLGetData(SQLHSTMT        StatementHandle,
            SQLUSMALLINT   Col_or_Param_Num,
            SQLSMALLINT     TargetType,
            SQLPOINTER      TargetValuePtr,
```

```
        SQLLEN        BufferLength,
        SQLLEN        *StrLen_or_IndPtr);
```

Parameter

Table 9-36 SQLGetData parameters

Keyword	Description
StatementHandle	Statement handle, obtained from **SQLAllocHandle**.
Col_or_Param_Num	Column number for which the data retrieval is requested. The column number starts with 1 and increases in ascending order. The number of the bookmark column is 0.
TargetType	C data type in the TargetValuePtr buffer. If **TargetType** is **SQL_ARD_TYPE**, the driver uses the data type of the **SQL_DESC_CONCISE_TYPE** field in ARD. If **TargetType** is **SQL_C_DEFAULT**, the driver selects a default data type according to the source SQL data type.
TargetValuePtr	**Output parameter**: pointer to the pointer that points to the buffer where the data is located.
BufferLength	Size of the buffer pointed to by **TargetValuePtr**.
StrLen_or_IndPtr	**Output parameter**: pointer to the buffer where the length or identifier value is returned.

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.

- **SQL_SUCCESS_WITH_INFO** indicates some warning information is displayed.

- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.

- **SQL_NO_DATA** indicates that the SQL statement does not return a result set.

- **SQL_INVALID_HANDLE** indicates that invalid handles were called. Values returned by other APIs are similar to the preceding values.

- **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. The **SQLSTATE** value provides the detailed function calling information.

Examples

See **Examples**.

9.4.5.19 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);
```

Parameter

Table 9-37 SQLGetDiagRec parameters

Keyword	Description
HandleType	A handle-type identifier that describes the type of handle for which diagnostics are desired. The value must be one of the following: • SQL_HANDLE_ENV • SQL_HANDLE_DBC • SQL_HANDLE_STMT • SQL_HANDLE_DESC
Handle	A handle for the diagnostic data structure. Its type is indicated by HandleType. If **HandleType** is **SQL_HANDLE_ENV**, **Handle** may be shared or non-shared environment handle.
RecNumber	Indicates the status record from which the application seeks information. RecNumber starts with 1.
SQLState	**Output parameter**: pointer to a buffer that saves the 5-character **SQLSTATE** code pertaining to **RecNumber**.
NativeErrorPtr	**Output parameter**: pointer to a buffer that saves the native error code.
MessageText	Pointer to a buffer that saves text strings of diagnostic information.
BufferLength	Length of MessageText.

Keyword	Description
TextLengthPtr	**Output parameter**: pointer to the buffer, the total number of bytes in the returned **MessageText**. If the number of bytes available to return is greater than **BufferLength**, then the diagnostics information text in **MessageText** is truncated to **BufferLength** minus the length of the null termination character.

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.

- **SQL_SUCCESS_WITH_INFO** indicates some warning information is displayed.

- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.

- **SQL_INVALID_HANDLE** indicates that invalid handles were called. Values returned by other APIs are similar to the preceding values.

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. No diagnostic records are generated.

- **SQL_INVALID_HANDLE**: The handle indicated by **HandType** and **Handle** is not a valid handle.

- **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 9-38 SQLSTATE values

SQLSATATE	Error	Description
HY000	General error	An error occurred for which there is no specific SQLSTATE.
HY001	Memory allocation error	The driver is unable to allocate memory required to support execution or completion of the function.

SQLSATATE	Error	Description
HY008	Operation canceled	SQLCancel is called to terminate the statement execution, but the StatementHandle function is still called.
HY010	Function sequence error	The function is called prior to sending data to data parameters or columns being executed.
HY013	Memory management error	The function fails to be called. The error may be caused by low memory conditions.
HYT01	Connection timed out	The timeout period expired before the application was able to connect to the data source.
IM001	Function not supported by the driver	The called function is not supported by the StatementHandle driver.

Examples

See **Examples**.

9.4.5.20 SQLSetConnectAttr

Function

SQLSetConnectAttr sets connection attributes.

Prototype

```
SQLRETURN SQLSetConnectAttr(SQLHDBC        ConnectionHandle
                SQLINTEGER    Attribute,
                SQLPOINTER    ValuePtr,
                SQLINTEGER    StringLength);
```

Parameter

Table 9-39 SQLSetConnectAttr parameters

Keyword	Description
StatementtHandle	Connection handle.
Attribute	Attribute to set.

Keyword	Description
ValuePtr	Pointer to the Attribute value. **ValuePtr** depends on the Attribute value, and can be a 32-bit unsigned integer value or a null-terminated string. If **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 succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. Values returned by other APIs are similar to the preceding values.

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. The **SQLSTATE** value provides the detailed function calling information.

Examples

See **Examples**.

9.4.5.21 SQLSetEnvAttr

Function

SQLSetEnvAttr sets environment attributes.

Prototype

```
SQLRETURN SQLSetEnvAttr(SQLHENV      EnvironmentHandle
             SQLINTEGER   Attribute,
             SQLPOINTER   ValuePtr,
             SQLINTEGER   StringLength);
```

Parameters

Table 9-40 SQLSetEnvAttr parameters

Keyword	Description
EnviromentHandle	Environment handle.
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 Attribute value. **ValuePtr** depends on the Attribute value, and can be a 32-bit integer value or a null-terminated string.
StringLength	If **ValuePtr** points to a string or a binary buffer, 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 succeeded.

- **SQL_SUCCESS_WITH_INFO** indicates some warning information is displayed.

- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.

- **SQL_INVALID_HANDLE** indicates that invalid handles were called. Values returned by other APIs are similar to the preceding values.

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. The **SQLSTATE** value provides the detailed function calling information.

Examples

See **Examples**.

9.4.5.22 SQLSetStmtAttr

Function

SQLSetStmtAttr sets attributes related to a statement.

Prototype

```
SQLRETURN SQLSetStmtAttr(SQLHSTMT       StatementHandle
                 SQLINTEGER   Attribute,
                 SQLPOINTER   ValuePtr,
                 SQLINTEGER   StringLength);
```

Parameter

Table 9-41 SQLSetStmtAttr parameters

Keyword	Description
StatementtHandle	Statement handle.
Attribute	Attribute to set.
ValuePtr	Pointer to the Attribute value. **ValuePtr** depends on the Attribute value, and can be a 32-bit unsigned integer value or a pointer to a null-terminated string, a binary buffer, 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 succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. Values returned by other APIs are similar to the preceding values.

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. The **SQLSTATE** value provides the detailed function calling information.

Examples

See **Examples**.

10 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.

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_NotSameAlignmentReason, 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)

This document contains open source software notice for the product. And this document is confidential information of copyright holder. Recipient shall protect it in due care and shall not disseminate it without permission.

Warranty Disclaimer

This document is provided "as is" without any warranty whatsoever, including the accuracy or comprehensiveness. Copyright holder of this document may change the contents of this document at any time without prior notice, and copyright holder disclaims any liability in relation to recipient's use of this document.

Open source software is provided by the author "as is" and any express or implied warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose are disclaimed. In no event shall the author be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of data or profits; or business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of open source software, even if advised of the possibility of such damage.

Copyright Notice And License Texts

Software: postgis-2.4.2

Copyright notice:

"Copyright (C) 1996-2015 Free Software Foundation, Inc.

Copyright (C) 1989, 1991 Free Software Foundation, Inc.,

51 Franklin Street, Fifth Floor, Boston, MA 02110-1301

Copyright 2008 Kevin Neufeld

Copyright (c) 2009 Walter Bruce Sinclair

Copyright 2006-2013 Stephen Woodbridge.

Copyright (c) 2008 Walter Bruce Sinclair

Copyright (c) 2012 TJ Holowaychuk <tj@vision-media.ca>

Copyright (c) 2008, by Attractive Chaos <attractivechaos@aol.co.uk>

Copyright (c) 2001-2012 Walter Bruce Sinclair

Copyright (c) 2010 Walter Bruce Sinclair

Copyright 2006 Stephen Woodbridge

Copyright 2006-2010 Stephen Woodbridge.

Copyright (c) 2006-2014 Stephen Woodbridge.

Copyright (c) 2017, Even Rouault <even.rouault at spatialys.com>

Copyright (C) 2004-2015 Sandro Santilli <strk@kbt.io>

Copyright (C) 2008-2011 Paul Ramsey <pramsey@cleverelephant.ca>

Copyright (C) 2008 Mark Cave-Ayland <mark.cave-ayland@siriusit.co.uk>

Copyright 2015 Nicklas Avén <nicklas.aven@jordogskog.no>

Copyright 2008 Paul Ramsey

Copyright (C) 2012 Sandro Santilli <strk@kbt.io>

Copyright 2012 Sandro Santilli <strk@kbt.io>

Copyright (C) 2014 Sandro Santilli <strk@kbt.io>

Copyright 2013 Olivier Courtin <olivier.courtin@oslandia.com>

Copyright 2009 Paul Ramsey <pramsey@cleverelephant.ca>

Copyright 2008 Paul Ramsey <pramsey@cleverelephant.ca>

Copyright 2011 Sandro Santilli <strk@kbt.io>

Copyright 2015 Daniel Baston

Copyright 2009 Olivier Courtin <olivier.courtin@oslandia.com>

Copyright 2014 Kashif Rasul <kashif.rasul@gmail.com> and

Shoaib Burq <saburq@gmail.com>

Copyright 2013 Sandro Santilli <strk@kbt.io>

Copyright 2010 Paul Ramsey <pramsey@cleverelephant.ca>

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>

Copyright (C) 2011 OpenGeo.org

Copyright (c) 2003-2017, Troy D. Hanson http:troydhanson.github.com/uthash/

Copyright (C) 2011 Regents of the University of California

Copyright (C) 2011-2013 Regents of the University of California

Copyright (C) 2010-2011 Jorge Arevalo <jorge.arevalo@deimos-space.com>

Copyright (C) 2010-2011 David Zwarg <dzwarg@azavea.com>

Copyright (C) 2009-2011 Pierre Racine <pierre.racine@sbf.ulaval.ca>

Copyright (C) 2009-2011 Mateusz Loskot <mateusz@loskot.net>

Copyright (C) 2008-2009 Sandro Santilli <strk@kbt.io>

Copyright (C) 2013 Nathaneil Hunter Clay <clay.nathaniel@gmail.com

Copyright (C) 2013 Nathaniel Hunter Clay <clay.nathaniel@gmail.com>

Copyright (C) 2013 Bborie Park <dustymugs@gmail.com>

Copyright (C) 2013 Nathaniel Hunter Clay <clay.nathaniel@gmail.com>

(C) 2009 Mateusz Loskot <mateusz@loskot.net>

Copyright (C) 2009 Mateusz Loskot <mateusz@loskot.net>

Copyright (C) 2009-2010 Mateusz Loskot <mateusz@loskot.net>

Copyright (C) 2009-2010 Jorge Arevalo <jorge.arevalo@deimos-space.com>

Copyright (C) 2012 Regents of the University of California

Copyright (C) 2013 Regents of the University of California

Copyright (C) 2012-2013 Regents of the University of California

Copyright (C) 2009 Sandro Santilli <strk@kbt.io>

"

License: The GPL v2 License.

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.

51 Franklin St, 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.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the

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 licensee 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

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 Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, 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

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "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 PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. 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 PROGRAM 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 PROGRAM (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 PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), 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 Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. 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.

Copyright (C) 2006 Refractions Research Inc.

Copyright (C) 2013 Sandro Santilli <strk@keybit.net>

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

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) 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) 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>

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) 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>

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) 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>

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) 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>

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) 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>

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) 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>

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) 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>

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.

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) 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>

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) 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>

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) 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>

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) 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>

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) 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) 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) 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.

[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 2011, John Resig

Copyright 2011, The Dojo Foundation

Copyright (c) 2012 Eric Haszlakiewicz

Copyright (c) 2009-2012 Hewlett-Packard Development Company, L.P.

Copyright (c) 2008-2009 Yahoo! Inc. All rights reserved.

Copyright (C) 1996, 1997, 1998, 1999, 2000, 2001, 2003, 2004, 2005, 2006,

2007, 2008, 2009, 2010, 2011 Free Software Foundation, Inc.

Copyright (c) 2013 Metaparadigm Pte. Ltd.

License: MIT License

Copyright (c) 2009-2012 Eric Haszlakiewicz

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Copyright (c) 2004, 2005 Metaparadigm Pte Ltd

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

Software: proj

Copyright notice:

"Copyright (C) 2010 Mateusz Loskot <mateusz@loskot.net>

Copyright (C) 2007 Douglas Gregor <doug.gregor@gmail.com>

Copyright (C) 2007 Troy Straszheim

CMake, Copyright (C) 2009-2010 Mateusz Loskot <mateusz@loskot.net>)

Copyright (C) 2011 Nicolas David <nicolas.david@ign.fr>

Copyright (c) 2000, Frank Warmerdam

Copyright (c) 2011, Open Geospatial Consortium, Inc.

Copyright (C) 1996, 1997, 1998, 1999, 2000, 2001, 2003, 2004, 2005, 2006,

2007, 2008, 2009, 2010, 2011 Free Software Foundation, Inc.

Copyright (c) Charles Karney (2012-2015) <charles@karney.com> and licensed

Copyright (c) 2005, Antonello Andrea

Copyright (c) 2010, Frank Warmerdam

Copyright (c) 1995, Gerald Evenden

Copyright (c) 2000, Frank Warmerdam <warmerdam@pobox.com>

Copyright (c) 2010, Frank Warmerdam <warmerdam@pobox.com>

Copyright (c) 2013, Frank Warmerdam

Copyright (c) 2003 Gerald I. Evenden

Copyright (c) 2012, Frank Warmerdam <warmerdam@pobox.com>

Copyright (c) 2002, Frank Warmerdam

Copyright (c) 2004 Gerald I. Evenden

Copyright (c) 2012 Martin Raspaud

Copyright (c) 2001, Thomas Flemming, tf@ttqv.com

Copyright (c) 2002, Frank Warmerdam <warmerdam@pobox.com>

Copyright (c) 2009, Frank Warmerdam

Copyright (c) 2003, 2006 Gerald I. Evenden

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 Resource Monitoring

GaussDB(DWS) provides multiple dimensional resource monitoring views to show the real-time and historical resource usage of tasks.

11.1 User Resource Monitoring

Context

In the multi-tenant management framework, you can query the real-time or historical usage of all user resources (including memory, CPU cores, storage space, temporary space, and I/Os).

📖 **NOTE**

- User real-time resource views/functions: **PG_TOTAL_USER_RESOURCE_INFO** and GS_WLM_USER_RESOURCE_INFO; user historical resource table: **GS_WLM_USER_RESOURCE_HISTORY**.

- You can monitor 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 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**.

11.2 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**.

11.3 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
```

To enable memory management, you can set **enable_memory_limit** and
max_process_memory on the GaussDB(DWS) console 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**.

11.4 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)
```

11.5 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 11-1 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**/**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.

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 11-2 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

11.6 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 flushed 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 (Database Manager interface)	**GS_WLM_SESSION_HISTORY**
		History (internal dump interface)	**GS_WLM_SESSION_INFO**
	All CNs	History (Database Manager interface)	**PGXC_WLM_SESSION_HISTORY**
		History (internal dump interface)	**PGXC_WLM_SESSION_INFO**
Operator level	Current CN	History (Database Manager interface)	**GS_WLM_OPERATOR_HISTORY**
		History (internal dump interface)	**GS_WLM_OPERAROR_INFO**
	All CNs	History (Database Manager interface)	**PGXC_WLM_OPERATOR_HISTORY**
		History (internal dump interface)	**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 11-2**.
- 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**.
- Job monitoring rules are as follows:
  - Jobs whose execution time recorded in the real-time resource monitoring view (see **Table 11-1**) is greater than or equal to **resource_track_duration**.
- 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.

11.7 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 11-1 EXPLAIN result

In this example, to demonstrate the resource monitoring function of top SQLs, 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

12 User-Defined Functions

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

12.1 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 12-1** 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.

First store the JAR package on an OBS server. For details, see "Uploading a File" in *Object Storage Service Console Operation Guide*. Then, create the access key AK/SK. For details about how to create access keys, see "Creating an Access Key (AK and SK)" in *Data Warehouse Service User Guide*. After that, log in to the database, run the gs_extend_library function, and import the package 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 12-1**.
- 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 | STATBLE | 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 12-1 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.

- **CHAR**, **NAME**, **BIGINT**, **INTEGER**, **SMALLINT**, **TINYINT**, **RAW**, **FLOAT4**, **FLOAT8**, **BPCHAR**, **VARCHAR**, **NVARCHAR2**, **DATE**, **TIMESTAMP**, **TIMESTAMPTZ**, **NUMERIC**, and **SMALLDATETIME**: They can be converted to the **TEXT** type by default.

- **TEXT**, **CHAR**, **BIGINT**, **INTEGER**, **SMALLINT**, **TINYINT**, **RAW**, **FLOAT4**, **FLOAT8**, **BPCHAR**, **DATE**, **NVARCHAR2**, **TIMESTAMP**, **NUMERIC**, and **SMALLDATETIME**: They can be converted to the **VARCHAR** type by default.

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

- **pljava_vmoptions**

  A session-level GUC parameter. It is used to set JVM startup parameters.

  ```
  SET pljava_vmoptions='-Xmx64m –Xms2m –XX:MaxMetaspaceSize=8m';
  ```

  pljava_vmoptions supports:

  – JDK8 JVM startup parameters.

  – JDK8 JVM system attributes (starting with **–D**, for example, **–Djava.ext.dirs**).

  > **NOTICE**
  >
  > You are not advised to set any parameters that contain directories because such setting may lead to unpredictable behavior.

  – User-defined parameters (starting with **–D**, for example, **–Duser.defined.option**).

  > **NOTICE**
  >
  > If a user sets **pljava_vmoptions** to a value beyond the value range, an error will be reported during function revoking.
  >
  > ```
  > SET pljava_vmoptions=' illegal.option';
  > SET
  > SELECT java_dummy(5::int);
  > ERROR:  UDF Error:cannot use PL/Java before successfully completing its setup.Please check if your pljava_vmoption is set correctly,since we do not ignore illegal parameters.Or check the log for more messages.
  > ```

- **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 12-2 PL/Java log levels

java.util.logging.Level	GaussDB(DWS) Log Level
SERVER	ERROR
WARNING	WARNING
CONFIG	LOG
INFO	INFO
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.

12.2 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 **Stored Procedures**. There is only one difference. Stored procedures have no return values but the functions have.

13 Stored Procedures

13.1 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, enabling business rule code to be moved from applications to databases and used by multiple programs at a time.

For details about how to create and call a stored procedure, see **CREATE PROCEDURE**.

The functions and stored procedures created by using PL/pgSQL in **PL/pgSQL Functions** are applicable to all the following sections.

13.2 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.

13.3 Data Type Conversion

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 13-1 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 13-1 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	-

13.4 Arrays and Records

13.4.1 Arrays

Use of Array Types

Before the use of arrays, an array type needs to be defined:

Define an array type immediately after the **AS** keyword in a stored procedure. 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
```

13.4.2 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 13-1**.

Figure 13-1 Syntax of the record type
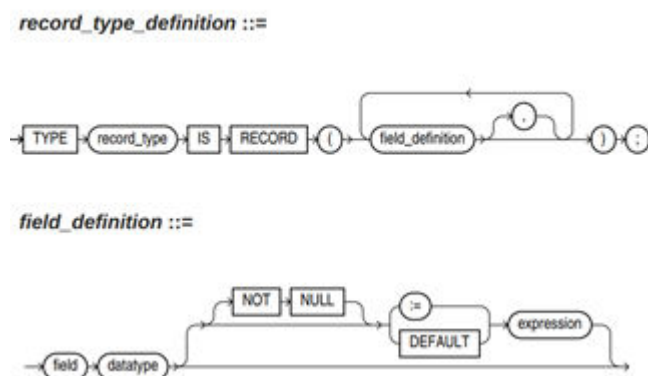
The syntax is described as follows:

- **record_type**: record name
- **field**: record columns
- **datatype**: record data type
- **expression**: expression for setting a default value

 NOTE

In GaussDB(DWS):

- 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;
```

13.5 Syntax

13.5.1 Basic Structure

Structure

A PL/SQL block can contain a sub-block which can be placed in any section. The following describes the architecture of a PL/SQL block:

- **DECLARE**: declares variables, types, cursors, and regional stored procedures and functions used in the PL/SQL block.
  ```
  DECLARE
  ```

> **NOTE**
>
> This part is optional if no variable needs to be declared.
>
> - An anonymous block may omit the **DECLARE** keyword if no variable needs to be declared.
> - For a stored procedure, **AS** is used, which is equivalent to **DECLARE**. The **AS** keyword must be reserved even if there is no variable declaration part.

- **EXECUTION**: specifies procedure and SQL statements. It is the main part of a program. Mandatory
  ```
  BEGIN
  ```
- **EXCEPTION**: processes errors. Optional
  ```
  EXCEPTION
  ```
- **END**
  ```
  END;
  /
  ```

> **NOTICE**
>
> You are not allowed to use consecutive tabs in the PL/SQL block, because they may result in an exception when the parameter **-r** is executed using the **gsql** tool.

Type

PL/SQL blocks are classified into the following types:

- Anonymous block: a dynamic block that can be executed only for once. For details about the syntax, see **Figure 13-2**.
- Subprogram: a stored procedure, function, operator, or packages stored in a database. A subprogram created in a database can be called by other programs.

13.5.2 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 13-2 shows the syntax diagrams for an anonymous block.

Figure 13-2 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;
/
```

13.5.3 Subprogram

A subprogram stores stored procedures, functions, operators, and advanced packages. A subprogram created in a database can be called by other programs.

13.6 Basic Statements

13.6.1 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 details about the variable declaration syntax, see **Figure 13-3**.

Figure 13-3 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.

Example:

```
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.

Scope of a Variable

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.

Example:

```
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;
/
```

13.6.2 Assignment Statement

Syntax

Figure 13-4 shows the syntax diagram for assigning a value to a variable.

Figure 13-4 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; --Assignment
BEGIN
    emp_id := 5; --Assignment
    emp_id := 5*7784;
END;
/
```

13.6.3 Call Statement

Syntax

Figure 13-5 shows the syntax diagram for calling a clause.

Figure 13-5 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);  --Invoke 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;
```
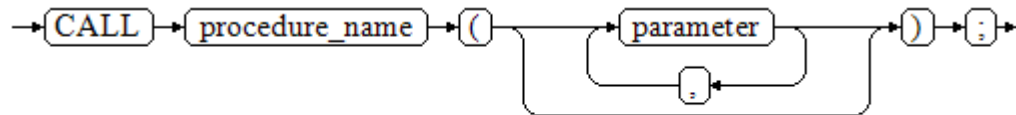
13.7 Dynamic Statements

13.7.1 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 13-6 shows the syntax diagram.

Figure 13-6 EXECUTE IMMEDIATE dynamic_select_clause::=
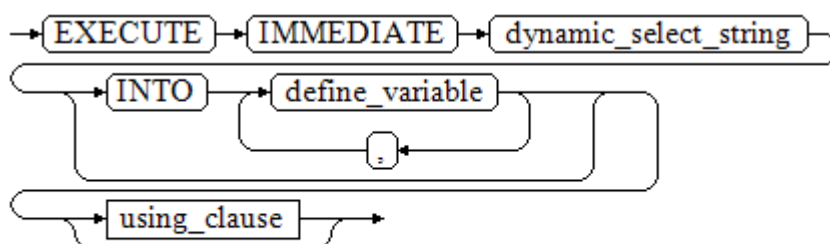
Figure 13-7 shows the syntax diagram for **using_clause**.

Figure 13-7 using_clause-1
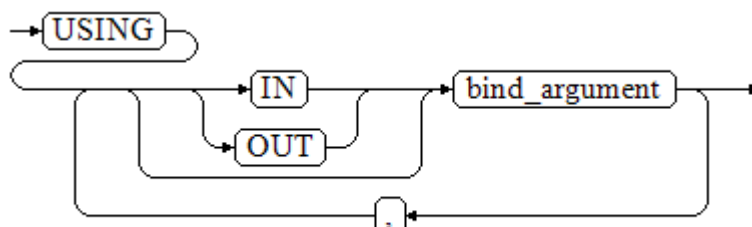
The above syntax diagram is explained as follows:

- **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 13-8**.

Figure 13-8 open_for::=
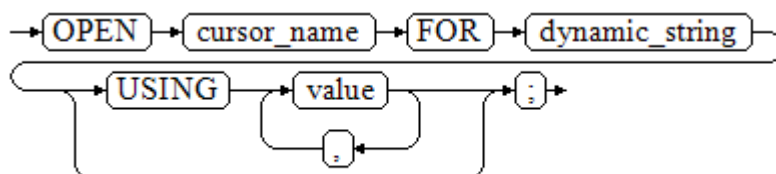
Parameter description:

- **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 **Cursors**.

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;
/
```

13.7.2 Executing Dynamic Non-query Statements

Syntax

Figure 13-9 shows the syntax diagram.

Figure 13-9 noselect::=
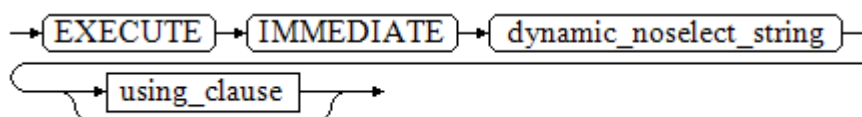
Figure 13-10 shows the syntax diagram for **using_clause**.

Figure 13-10 using_clause-2
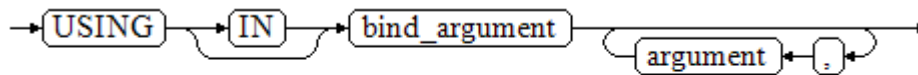
The above syntax diagram is explained as follows:

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;
```

13.7.3 Dynamically Calling Stored Procedures

This section describes how to dynamically call store procedures. You must use anonymous statement blocks to package stored procedures or statement blocks

and append **IN** and **OUT** behind the **EXECUTE IMMEDIATE...USING** statement to input and output parameters.

Syntax

Figure 13-11 shows the syntax diagram.

Figure 13-11 call_procedure::=
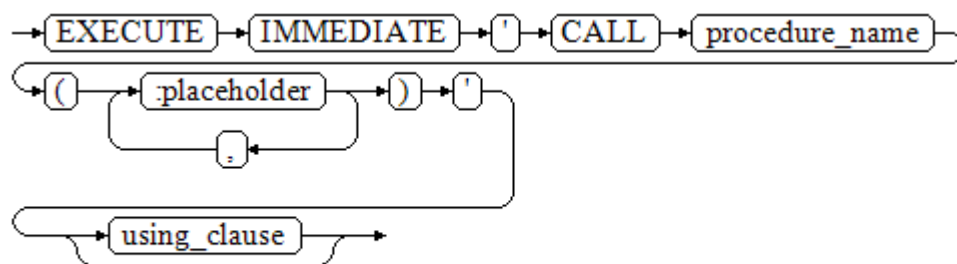
Figure 13-12 shows the syntax diagram for **using_clause**.

Figure 13-12 using_clause-3
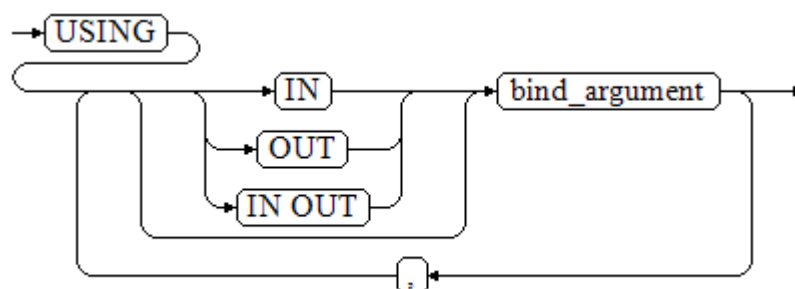
The above syntax diagram is explained as follows:

- **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;
```

13.7.4 Dynamically Calling Anonymous Blocks

This section describes how to execute anonymous blocks in dynamic statements. Append **IN** and **OUT** behind the **EXECUTE IMMEDIATE...USING** statement to input and output parameters.

Syntax

Figure 13-13 shows the syntax diagram.

Figure 13-13 call_anonymous_block::=
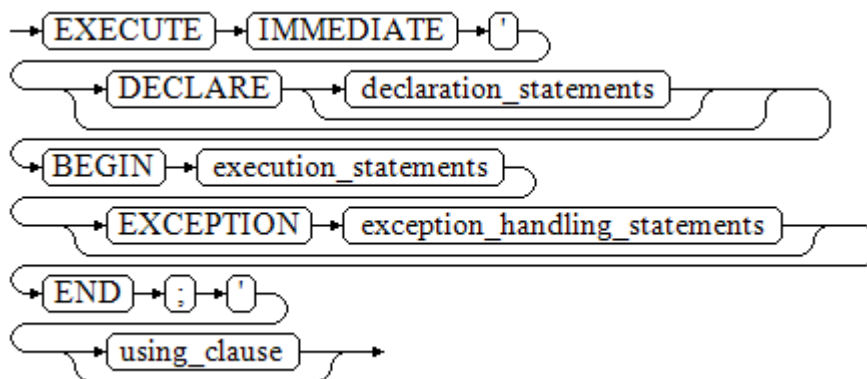
Figure 13-14 shows the syntax diagram for **using_clause**.

Figure 13-14 using_clause-4
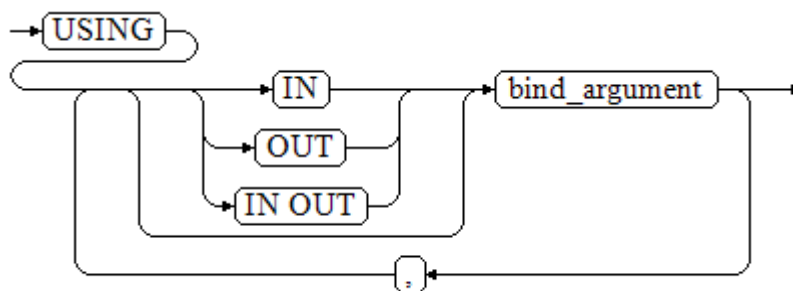
The above syntax diagram is explained as follows:

- 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;
```

13.8 Control Statements

13.8.1 RETURN Statements

In GaussDB(DWS), data can be returned in either of the following ways: **RETURN**, **RETURN NEXT**, or **RETURN QUERY**. **RETURN NEXT** and **RETURN QUERY** are used only for functions and cannot be used for stored procedures.

13.8.1.1 RETURN

Syntax

Figure 13-15 shows the syntax diagram for a return statement.

Figure 13-15 return_clause::=
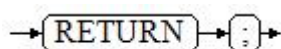
The syntax details are as follows:

This statement returns control from a stored procedure or function to a caller.

Examples

See **Examples** for call statement examples.

13.8.1.2 RETURN NEXT and RETURN QUERY

Syntax

When creating a function, specify **SETOF datatype** for the return values.

return_next_clause::=

return_query_clause::=
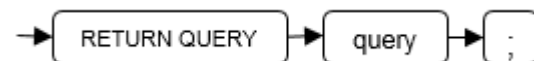
The syntax details are as follows:

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)
```

13.8.2 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 13-16 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 13-17 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 sex = 'm' THEN
    pretty_sex := 'man';
ELSE
    IF sex = 'f' THEN
        pretty_sex := 'woman';
    END IF;
END IF;
```

Actually, this is a way of an **IF** statement nesting in the **ELSE** part of another **IF** statement. Therefore, an **END IF** statement is required for each nesting **IF** statement and another **END IF** statement to end the parent **IF-ELSE** statement. To set multiple options, use the following form:

- IF_THEN_ELSIF_ELSE

Figure 13-18 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;
```

13.8.3 Loop Statements

Simple LOOP Statements

The syntax diagram is as follows.

Figure 13-19 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 13-20 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 13-21 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 13-22 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 13-23 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;
```

13.8.4 Branch Statements

Syntax

Figure 13-24 shows the syntax diagram.

Figure 13-24 case_when::=

Figure 13-25 shows the syntax diagram for **when_clause**.

Figure 13-25 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;
```

13.8.5 NULL Statements

In PL/SQL programs, **NULL** statements are used to indicate "nothing should be done", equal to placeholders. They grant meanings to some statements and improve program readability.

Syntax

The following shows example use of NULL statements.

```
DECLARE
    …
BEGIN
    …
    IF v_num IS NULL THEN
        NULL; --No data needs to be processed.
    END IF;
END;
/
```

13.8.6 Error Trapping Statements

By default, any error occurring in a PL/SQL function aborts execution of the function, and indeed of the surrounding transaction as well. You can trap errors and restore from them by using a **BEGIN** block with an **EXCEPTION** clause. The syntax is an extension of the normal syntax for a **BEGIN** block:

```
[<<label>>]
[DECLARE
    declarations]
BEGIN
    statements
EXCEPTION
    WHEN condition [OR condition …] THEN
        handler_statements
    [WHEN condition [OR condition …] THEN
        handler_statements
    …]
END;
```

If no error occurs, this form of block simply executes all the statements, and then control passes to the next statement after **END**. But if an error occurs within the statements, further processing of the statements is abandoned, and control passes to the **EXCEPTION** list. The list is searched for the first condition matching the error that occurred. If a match is found, the corresponding **handler_statements** are executed, and then control passes to the next statement after **END**. If no match is found, the error propagates out as though the **EXCEPTION** clause were not there at all:

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 ;
```

13.8.7 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 ::=
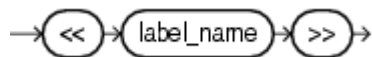
goto statement ::=
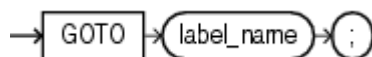
Examples

```
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;
  /
```

13.9 Other Statements

13.9.1 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.

13.9.2 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**.

13.10 Cursors

13.10.1 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 13-2** shows the usage conditions of explicit and implicit cursors for different SQL statements.

Table 13-2 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

13.10.2 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 13-26 shows the syntax diagram for defining a static cursor.

Figure 13-26 static_cursor_define::=
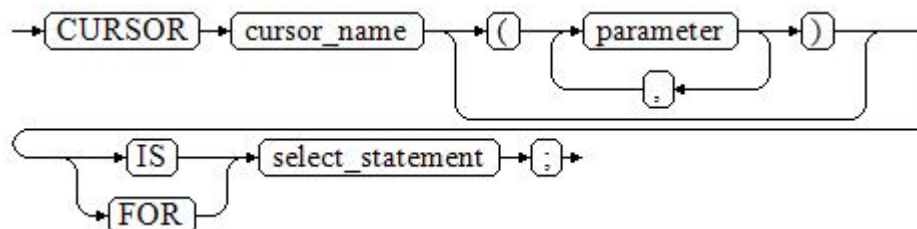
Parameter description:

- **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 13-27 and **Figure 13-28** show the syntax diagrams for defining a dynamic cursor.

Figure 13-27 cursor_typename::=
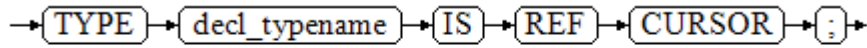
GaussDB(DWS) supports the dynamic cursor type **sys_refcursor**. A function or stored procedure can use the **sys_refcursor** parameter to pass on or pass out the cursor result set. A function can return **sys_refcursor** to return the cursor result set.

Figure 13-28 dynamic_cursor_define::=
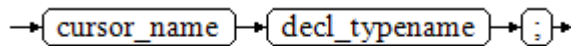
Step 2 **Open the static cursor:** Execute the **SELECT** statement corresponding to the cursor. The query result is placed in the work area and the pointer directs to the head of the work area to identify the cursor result set. If the cursor query statement contains the **FOR UPDATE** option, the **OPEN** statement locks the data row corresponding to the cursor result set in the database table.

Figure 13-29 shows the syntax diagram for opening a static cursor.

Figure 13-29 open_static_cursor::=
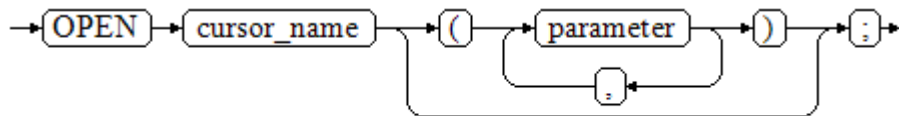
Open the dynamic cursor: Use the **OPEN FOR** statement to open the dynamic cursor and the SQL statement is dynamically bound.

Figure 13-30 shows the syntax diagram for opening a dynamic cursor.

Figure 13-30 open_dynamic_cursor::=
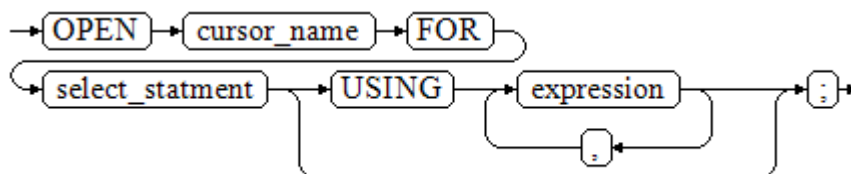
A PL/SQL program cannot use the **OPEN** statement to repeatedly open a cursor.

Step 3 **Fetch cursor data**: Retrieve data rows in the result set and place them in specified output variables.

Figure 13-31 shows the syntax diagram for fetching cursor data.

Figure 13-31 fetch_cursor::=
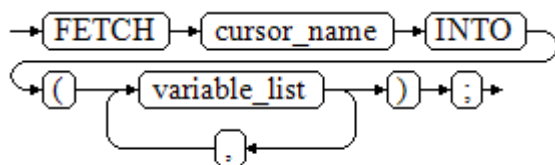
Step 4 Process the record.

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 13-32 shows the syntax diagram for closing a cursor.

Figure 13-32 close_cursor::=
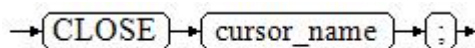
----End

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;
```

13.10.3 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;
```

13.10.4 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 13-33 shows the syntax diagram for the **FOR AS** loop.

Figure 13-33 FOR_AS_loop::=
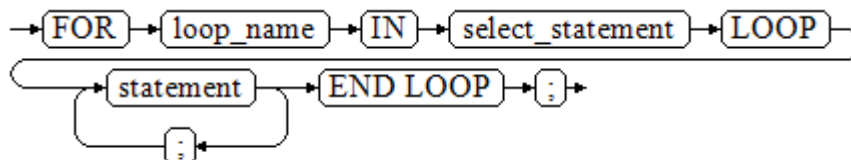
Precautions

- 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;
```

13.11 Advanced Packages

13.11.1 DBMS_LOB

Related Interfaces

Table 13-3 provides all interfaces supported by the **DBMS_LOB** package.

Table 13-3 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.

API	Description
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.
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 13-4 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 13-5 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 13-6 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 13-7 DBMS_LOB.WRITE 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.
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 13-8 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 13-9 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 13-10 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.

Parameter	Description
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 13-11 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 13-12 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.

Parameter	Description
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 13-13 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 13-14 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.
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 13-15 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 13-16 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 13-17 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;
```

13.11.2 DBMS_RANDOM

Related Interfaces

Table 13-18 provides all interfaces supported by the **DBMS_RANDOM** package.

Table 13-18 DBMS_RANDOM interface parameters

API	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 13-19 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 13-20 DBMS_RANDOM.VALUE interface parameters

Paramet er	Description
low	Sets the low bound for a random number. The generated random number is greater than or equal to the low.
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.

Examples

```
-- Generate a random number between 0 and 1:
SELECT DBMS_RANDOM.VALUE(0,1);

-- Add 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));
```

13.11.3 DBMS_OUTPUT

Related Interfaces

Table 13-21 provides all interfaces supported by the **DBMS_OUTPUT** package.

Table 13-21 DBMS_OUTPUT

API	Description
DBMS_OUTP UT.PUT_LINE	Outputs the specified text. The text length cannot exceed 32,767 bytes.
DBMS_OUTP UT.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.

API	Description
DBMS_OUTP UT.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 13-22 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 13-23 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 13-24 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;
/
```

13.11.4 UTL_RAW

Related Interfaces

Table 13-25 provides all interfaces supported by the **UTL_RAW** package.

Table 13-25 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 13-26 UTL_RAW.CAST_FROM_BINARY_INTEGER interface parameters

Parameter	Description
n	Specifies the INTEGER type value to be converted to the RAW type.

Paramete r	Description
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 13-27 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 13-28 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 13-29 UTL_RAW.CAST_TO_RAW interface parameters

Parameter	Description
c	Specifies a VARCHAR2 type object to be converted.

Examples

```
-- 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;
/

-- Invoke the stored procedure.
CALL proc_raw();

-- Delete the stored procedure.
DROP PROCEDURE proc_raw;
```

13.11.5 DBMS_JOB

Related Interfaces

Table 13-30 lists all interfaces supported by the **DBMS_JOB** package.

Table 13-30 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.

Interface	Description
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 13-31 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.

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
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 13-32 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 13-33 DBMS_JOB.BROKEN interface parameters

Param eter	Type	Input/ Outpu t Param eter	Ca n Be Em pty	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_da te	timesta mp	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 13-34 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 13-35 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 13-36 DBMS_JOB.NEXT_DATE interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
job	integer	IN	No	Specifies the job number.
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 13-37 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 13-38 DBMS_JOB.CHANGE_OWNER interface parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
job	integer	IN	No	Specifies the job number.

Paramet er	Type	Input/ Output Paramet er	Can Be Empty	Description
new_own er	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.

13.11.6 DBMS_SQL

Related Interfaces

Table 13-39 lists interfaces supported by the **DBMS_SQL** package.

Table 13-39 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.FETCH_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.
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.

API	Description
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 13-40 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 13-41 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 13-42 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 13-43 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 13-44 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 13-45 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 13-46 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 13-47 DBMS_SQL.DEFINE_COLUMN_LONG 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_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 13-48 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 13-49 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
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 13-50 DBMS_SQL.DEFINE_COLUMN_UNKNOWN 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
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 13-51 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 13-52 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

Parameter Name	Description
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 13-53 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 13-54 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

Parameter Name	Description
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 13-55 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 13-56 DBMS_SQL.COLUMN_VALUE_TEXT 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.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 13-57 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 13-58 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;
```

13.12 Debugging

Syntax

RAISE has the following five syntax formats:

Figure 13-34 raise_format::=
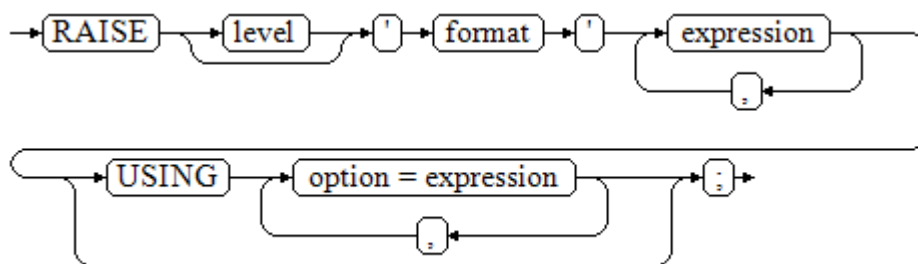
Figure 13-35 raise_condition::=
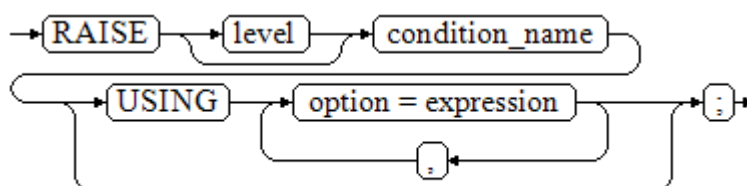
Figure 13-36 raise_sqlstate::=
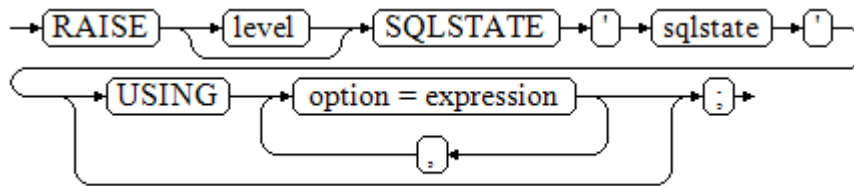
Figure 13-37 raise_option::=
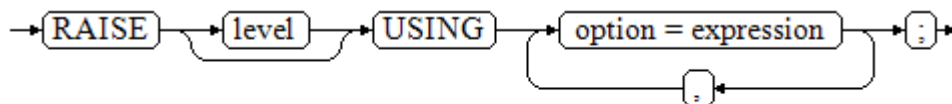
Figure 13-38 raise::=
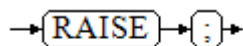
Parameter description:

- 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**. Do not use an error code ended with three zeros because this kind of error codes are type codes and can be captured by the whole category.

> 📖 **NOTE**
>
> The syntax described in **Figure 13-38** 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);

-- Execution result:
ERROR:  Noexistence ID --> 300011
HINT:  Please check your user ID
```

Two methods are available for setting **SQLSTATE**:

```
CREATE OR REPLACE PROCEDURE proc_raise2(user_id in integer)
AS
BEGIN
RAISE 'Duplicate user ID: %',user_id USING ERRCODE = 'unique_violation';
END;
/

\set VERBOSITY verbose
call proc_raise2(300011);

-- Execution result:
ERROR:  Duplicate user ID: 300011
SQLSTATE: 23505
LOCATION:  exec_stmt_raise, pl_exec.cpp:3482
```

If the main parameter is a condition name or **SQLSTATE**, the following applies:

RAISE division_by_zero;

RAISE SQLSTATE '22012';

For example:

```
CREATE OR REPLACE PROCEDURE division(div in integer, dividend in integer)
AS
DECLARE
```

```
      res int;
    BEGIN
    IF dividend=0 THEN
        RAISE division_by_zero;
        RETURN;
    ELSE
        res := div/dividend;
        RAISE INFO 'division result: %', res;
        RETURN;
    END IF;
    END;
/
call division(3,0);

-- Execution result:
ERROR:  division_by_zero
```

Alternatively:

```
RAISE unique_violation USING MESSAGE = 'Duplicate user ID: ' || user_id;
```

14 System Catalogs and System 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.

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 14-1 GS_OBSSCANINFO columns

Name	Type	Reference	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_st am	-	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 14-2 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

Name	Type	Description
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.
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.

Name	Type	Description
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.
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 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 14-3 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.

Name	Type	Description
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).
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**.
- This system catalog has a distribution column, the gaussdb column, in PostgreSQL databases only, not other databases.
- The **pg_catalog** has the **GS_WLM_OPERATOR_INFO** view.

Table 14-4 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
min_spill_size	integer	Minimum spilled data among all DNs when a spill occurs. The unit is MB. The default value is **0**.

Name	Type	Description
max_spill_size	integer	Maximum spilled data among all DNs when a spill occurs. The unit is MB. The default value is **0**.
average_spill_size	integer	Average spilled data among all DNs when a spill occurs. The unit is MB. The default value is **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 14-127**.

📖 **NOTE**

- This system catalog's schema is **dbms_om**.
- This system catalog has a distribution column, the gaussdb column, in PostgreSQL databases only, not other databases.
- 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 14-5 GS_WLM_USER_RESOURCE_HISTORY column

Name	Type	Description
username	text	Username
timestamp	timestamp with time zone	Timestamp
used_memory	int	Specifies the used memory (unit: MB)
total_memory	int	Available memory (unit: MB). **0** indicates that the available memory is not limited and depends on the maximum memory available in the database.
used_cpu	real	Number of CPU cores in use
total_cpu	int	Total number of CPU cores of the Cgroup associated with a user on the node
used_space	bigint	Used storage space (unit: KB)
total_space	bigint	Available storage space (unit: KB). **-1** indicates that the storage space is not limited.
used_temp_space	bigint	Used temporary storage space (unit: KB)
total_temp_space	bigint	Available temporary storage space (unit: KB). **-1** indicates that the maximum temporary storage space is not limited.
used_spill_space	bigint	Used space of operator flushing (unit: KB)
total_spill_space	bigint	Available storage space for operator flushing (unit: KB). The value **-1** indicates that the maximum operator flushing space is not limited.
read_kbytes	bigint	Byte traffic of read operations in a monitoring period (unit: KB)

Name	Type	Description
write_kbytes	bigint	Byte traffic of write operations in a monitoring period (unit: 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 (unit: KB)
write_speed	real	Byte rate of write operations in a monitoring period (unit: 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 14-6 PG_AGGREGATE columns

Name	Type	Reference	Description
aggfnoid	regproc	**PG_PROC**.oid	**PG_PROC** OID of the aggregate function
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.

Name	Type	Reference	Description
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 14-7 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
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

Name	Type	Reference	Description
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
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

Name	Type	Reference	Description
amvacuumclean up	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 14-8 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
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

Name	Type	Reference	Description
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 14-9 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
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 **amprocleftype** and **amprocrighttype** both equal to the index opclass's **opcintype**.

14.2.11 PG_ATTRDEF

PG_ATTRDEF stores default values of columns.

Table 14-10 PG_ATTRDEF columns

Name	Type	Description
adrelid	oid	Table to which the column belongs
adnum	smallint	Number of the 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 14-11 PG_ATTRIBUTE columns

Name	Type	Description
attrelid	oid	Table to which the column belongs
attname	name	Column name
atttypid	oid	Column type
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.

Name	Type	Description
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.
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

Name	Type	Description
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.

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 14-12 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

Column	Type	Description
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	For roles that can log in, this sets maximum number of concurrent connections this role can make. -1 means no limit.
rolpassword	text	Password (possibly encrypted); **NULL** if no password.
rolvalidbegin	timestamp with time zone	Account validity start time; **NULL** if no start time
rolvaliduntil	timestamp with time zone	Password expiry time; **NULL** if no expiration
rolrespool	name	Resource pool that a user can use
roluseft	boolean	Whether the role can perform operations on foreign tables
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

Column	Type	Description
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 14-13 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 14-14 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

Name	Type	Description
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 14-15 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.
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 14-16 PG_CLASS columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute; must be explicitly selected)

Name	Type	Description
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**.
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

Name	Type	Description
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.
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. See **PG_TRIGGER**.

Name	Type	Description
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
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.

Name	Type	Description

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 14-17**.

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 14-17 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

Parameter	Description
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%';
```

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 14-18 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

Name	Type	Reference	Description
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 14-19 PG_CONSTRAINT columns

Name	Type	Description
conname	name	Constraint name (not necessarily unique)
connamespace	oid	OID of the namespace that contains the constraint
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**.

Name	Type	Description
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).
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

Name	Type	Description
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's 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.

14.2.20 PG_CONVERSION

PG_CONVERSION records encoding conversion information.

Table 14-20 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 14-21 PG_DATABASE columns

Name	Type	Description
datname	name	Database name
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
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*';

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 14-22 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 14-23 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 14-24 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 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 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 14-25 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 14-26 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 12 extensions, that is, PLPGSQL, DIST_FDW, FILE_FDW, HDFS_FDW, HSTORE, PLDBGAPI, DIMSEARCH, PACKAGES, GC_FDW, UUID-OSSP, LOG_FDW, and ROACH_API.

Table 14-27 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 14-28 PG_EXTENSION_DATA_SOURCE columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute; must be explicitly selected)
srcname	name	-	Name of an external data source

Name	Type	Reference	Description
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 14-29 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 14-30 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 14-31 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 options

14.2.32 PG_INDEX

PG_INDEX records part of the information about indexes. The rest is mostly in **PG_CLASS**.

Table 14-32 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 14-33 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 14-34 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	Job invalid or not. If a job fails to be executed for 16 consecutive times, **is_broken** is automatically set to **true** and the job will not be executed later.
start_date	timestamp without time zone	Start time of the first job execution, accurate to millisecond

Name	Type	Description
next_run_date	timestamp without time zone	Scheduled time of the next job execution, accurate to millisecond
failure_count	smallint	Number of times the job has started and failed. If a job fails to be executed for 16 consecutive times, no more attempt will be made on it.
interval	text	Job execution interval
last_start_date	timestamp without time zone	Start time of the last job execution, accurate to millisecond
last_end_date	timestamp without time zone	End time of the last job execution, accurate to millisecond
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 programming languages. You can use them and interfaces to write functions or stored procedures.

Table 14-35 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

Name	Type	Reference	Description
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.
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 14-36 PG_LARGEOBJECT columns

Name	Type	Reference	Description
loid	oid	**PG_LARGEOBJECT_ME TADATA**.oid	Identifier of the large object that includes this page
pageno	integer	-	Page number of this page within its large object (counting from zero)
data	bytea	-	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 14-37 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 14-38 PG_NAMESPACE columns

Name	Type	Description
nspname	name	Name of the namespace
nspowner	oid	Owner of the namespace

Name	Type	Description
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 14-39 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. **object_mtime_record_mode** can be used to control whether **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 14-40 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 14-41 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
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 14-42 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

Name	Type	Reference	Description
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
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 14-43 PG_OPFAMILY columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute; must be explicitly selected)

Name	Type	Reference	Description
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 14-44 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.

Name	Type	Description
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
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

Name	Type	Description
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)

14.2.45 PG_PLTEMPLATE

PG_PLTEMPLATE records template information for procedural languages.

Table 14-45 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 14-46 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	int2vect or	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)
```

14.2.47 PG_RANGE

PG_RANGE records information about range types.

This is in addition to the types' entries in **PG_TYPE**.

Table 14-47 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
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

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.

14.2.48 PG_REDACTION_COLUMN

PG_REDACTION_COLUMN records the information about the redacted columns.

Table 14-48 PG_REDACTION_COLUMN columns

Name	Type	Description
object_oid	oid	OID of the object to be redacted.

Name	Type	Description
column_attrno	smallint	**attrno** of the redacted column.
function_type	integer	Redaction type. **NOTE** This column is reserved. It is used only for forward compatibility of redacted column information in earlier versions. The value can be **0** (NONE) or **1** (FULL).
function_parameters	text	Parameters used when the redaction type is **partial** (reserved).
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).
column_description	text	Description of the redacted column.
function_expr	pg_node_tree	Internal representation of the redaction function.
inherited	bool	Whether a redacted column is inherited from another redacted column.

14.2.49 PG_REDACTION_POLICY

PG_REDACTION_POLICY records information about the object to be redacted.

Table 14-49 PG_REDACTION_POLICY columns

Name	Type	Description
object_oid	oid	OID of the object to be redacted.
policy_name	name	Name of the redact 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 table corresponds to a physical file on the disk and the size of the file.

Table 14-50 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**.
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**.

Name	Type	Description
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 14-51 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 the information about database resource pool.

Table 14-52 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	Value of cores bound to the CPU
control_group	name	Name of the Cgroup where the resource pool is located
active_statements	integer	Maximum number of concurrent statements in the resource pool
max_dop	integer	Maximum concurrency. This is a reserved parameter.
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.

14.2.53 PG_REWRITE

PG_REWRITE records rewrite rules defined for tables and views.

Table 14-53 PG_REWRITE columns

Name	Type	Description
rulename	name	Rule name
ev_class	oid	Name of the table that uses the rule
ev_attr	smallint	Column this rule is for (always **0** to indicate the entire table)

Name	Type	Description
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 14-54 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 attempting to delete them.

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 14-55 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**.
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 psql'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 14-56 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 14-57 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 14-58 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

Name	Type	Description
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
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 the extended statistics of tables in a database, such as statistics of multiple columns. Statistics of expressions will be supported later. You can specify the extended statistics to be collected. It is accessible only to users with system administrator rights.

Table 14-59 PG_STATISTIC_EXT columns

Parameter	Type	Description
starelid	oid	Table or index which the described column belongs to
starelkind	"char"	Type of an object

Parameter	Type	Description
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.
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

Parameter	Type	Description
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 14-60 PG_SYNONYM columns

Name	Type	Description
synname	name	Synonym name.
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 14-61 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.

Name	Type	Description
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
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 14-62 PG_TS_CONFIG columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute; must be explicitly selected)
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 14-63 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 14-64 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 14-65 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

Name	Type	Reference	Description
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
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 14-66 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 14-67 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
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.

Name	Type	Description
typrelid	oid	If this is a composite type (see **typtype**), then this column points to the **pg_class** entry that defines the corresponding table. For a free-standing composite type, the **pg_class** entry does not represent a table, but it is required for the type's **pg_attribute** entries to link to. The value is **0** for non-composite types.
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 14-68 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 14-69 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 14-70 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 14-71 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_ex tend	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 14-72 PGXC_GROUP columns

Name	Type	Description
group_name	name	Name of the node group
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 14-73 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 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 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 14-74 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.

14.2.76 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 14-75 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.

14.2.77 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 14-76 ALL_ALL_TABLES columns

Name	Type	Description
owner	name	Owner of the table or the 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 14-77 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)

Name	Type	Description
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 14-78 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 14-79 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 14-80 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 14-81 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 14-82 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 14-83 ALL_INDEXES columns

Name	Type	Description
owner	character varying(64)	Index owner
index_name	character varying(64)	Index name
table_name	character varying(64)	Name of 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 14-84 ALL_OBJECTS columns

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	ID of the namespace where the object resides
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 14-85 ALL_PROCEDURES columns

Name	Type	Description
owner	name	Owner of the object
object_name	name	Object name

14.3.11 ALL_SEQUENCES

ALL_SEQUENCES displays all sequences accessible to the current user.

Table 14-86 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** indicates that it is a cycle sequence. ● **N** indicates that 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 14-87 ALL_SOURCE columns

Name	Type	Description
owner	name	Owner of the object
name	name	Object name
type	name	Type of the object
text	text	Definition of the object

14.3.13 ALL_SYNONYMS

ALL_SYNONYMS displays all synonyms accessible to the current user.

Table 14-88 ALL_SYNONYMS columns

Name	Type	Description
owner	text	Owner of a synonym.

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.

14.3.14 ALL_TAB_COLUMNS

ALL_TAB_COLUMNS displays description of columns of the tables and views that the current user can access.

Table 14-89 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.
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.

Name	Type	Description
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 14-90 ALL_TAB_COMMENTS columns

Name	Type	Description
owner	character varying(64)	Owner of the table or the 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 14-91 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
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	The 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 14-92 ALL_USERS columns

Name	Type	Description
username	name	User name
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 14-93 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
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 14-94 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 14-95 DBA_USERS columns

Name	Type	Description
username	character varying(64)	User name

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

Name	Type	Description
index_name	character varying(64)	Index name
table_owner	character varying(64)	Table owner
table_name	character varying(64)	Table name
column_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
index_name	character varying(64)	Index name of the partitioned table to which the index partition belongs

Name	Type	Description
partition_nam e	character varying(64)	Name of the index partition
index_partitio n_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_v alue	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_tablespac e_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 14-96 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 e	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 the table or the 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

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

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	The 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 14-97 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 14-98 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 sequences of the columns in the view are the same as those in the **GS_COLUMN_TABLE_IO_STAT** view. For details about the columns, see **GS_COLUMN_TABLE_IO_STAT**. The value of each statistical column is the sum of the values of the corresponding columns of all nodes.

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.

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. 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 **GS_ROW_TABLE_IO_STAT**. The value of each statistical column is the sum of the values of the corresponding columns of all nodes.

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 14-99**.

Table 14-99 GLOBAL_STAT_DATABASE columns

Name	Type	Description	Sum Range
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.	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

Name	Type	Description	Sum Range
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
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

Name	Type	Description	Sum Range
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 14-100 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
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)

Name	Type	Description
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 14-101 GLOBAL_TABLE_STAT columns

Name	Type	Description
schemaname	name	Namespace of a table
relname	name	Table name
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.

Name	Type	Description
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 14-102 GLOBAL_WORKLOAD_SQL_COUNT columns

Name	Type	Description
workload	name	Workload Cgroup name
select_count	bigint	Number of **SELECT** statements

Name	Type	Description
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 14-103 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**
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**

Name	Type	Description
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 14-104 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
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 14-105 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 to which a Workload 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 14-106 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
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 14-107 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 14-108**. 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 14-108 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
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)

Name	Type	Description
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)
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).

Name	Type	Description
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)
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.

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 14-109 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 14-110 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 14-111 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.
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	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.
used_mem	int	Memory size used by resource pools, 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 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 14-112 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
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 14-113 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.

Name	Type	Description
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 14-114 GS_SESSION_MEMORY_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_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	Information about statement flushing into disks on DNs **None** indicates that the statement has not been flushed to disks on any DNs. **All** indicates that the statement has been flushed to disks on every DN. [*a*:*b*] indicates that the statement has been flushed 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 14-115 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

Name	Type	Description
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
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 14-116 GS_STAT_DB_CU columns

Name	Type	Description
node_name1	text	Node name
db_name	text	Database name

Name	Type	Description
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 14-117 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 hard disk synchronous reads
hdd_asyn_read	integer	Number of hard 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 14-118 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

Name	Type	Description
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

14.3.68 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 14-119 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.

Name	Type	Description
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.
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 14-120 GS_TOTAL_NODEGROUP_MEMORY_DETAIL columns

Name	Type	Description
ngname	text	Name of a logical cluster

Name	Type	Description
memorytype	text	Memory type. Its value can be: • **ng_total_memory**: total memory of the logical cluster • **ng_used_memory**: memory usage of the logical cluster • **ng_estimate_memory**: estimated memory usage of the logical cluster • **ng_foreignrp_memsize**: total memory of the external resource pool of the logical cluster • **ng_foreignrp_memsize**: memory usage of the external resource pool of the logical cluster • **ng_foreignrp_peaksize**: peak memory usage of the external resource pool of the logical cluster • **ng_foreignrp_mempct**: percentage of the external resource pool of the logical cluster to the total memory of the logical cluster • **ng_foreignrp_estmsize**: estimated memory usage of the external resource pool of the logical cluster
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 14-121 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 14-122 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 14-123 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 14-124 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 14-125 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.

Name	Type	Description
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**.

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 by Database Manager to query data from the kernel. Data in the kernel 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 14-4**.

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 14-126 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
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

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. The default value is **0**.
max_spill_size	integer	Maximum spilled data among all DNs when a spill occurs. The unit is MB. The default value is **0**.
average_spill_size	integer	Average spilled data among all DNs when a spill occurs. The unit is MB. The default value is **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

14.3.78 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**.

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 14-127 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.

Name	Type	Description
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 (unit: 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 flushed to disks on every DN. [*a*:*b*] indicates that the statement has been flushed 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. The default value is **0**.

Name	Type	Description
max_spill_size	integer	Maximum spilled data among all DNs when a spill occurs. The unit is MB. The default value is **0**.
average_spill_size	integer	Average spilled data among all DNs when a spill occurs. The unit is MB. The default value is **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 IOPS peak of a statement across all DNs. It is counted by ones in a column-store table and by ten thousands in a row-store table.
max_peak_iops	integer	Maximum IOPS peak of a statement across all DNs. It is counted by ones in a column-store table and by ten thousands in a row-store table.
average_peak_iops	integer	Average IOPS peak of a statement across all DNs. It is counted by ones in a column-store table and by ten thousands in a row-store table.
iops_skew_percent	integer	I/O skew across DNs.

Name	Type	Description
warning	text	Warning. The following warnings and warnings related to SQL self-diagnosis tuning are displayed: 1. Spill file size large than 256MB 2. Broadcast size large than 100MB 3. Early spill 4. Spill times is greater than 3 5. Spill on memory adaptive 6. 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 14-128 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
username	name	User name used for connecting to the backend
application_name	text	Name of the application that is connected to the backend

Name	Type	Description
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.
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.

Name	Type	Description
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 flushed to disks on every DN. **[*a*:*b*]** indicates that the statement has been flushed 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. The default value is **0**.
max_spill_size	integer	Maximum spilled data among all DNs when a spill occurs. The unit is MB. The default value is **0**.
average_spill_size	integer	Average spilled data among all DNs when a spill occurs. The unit is MB. The default value is **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 IOPS peak of a statement across all DNs. It is counted by ones in a column-store table and by ten thousands in a row-store table.

Name	Type	Description
max_peak_iops	integer	Maximum IOPS peak of a statement across all DNs. It is counted by ones in a column-store table and by ten thousands in a row-store table.
average_peak_iops	integer	Average IOPS peak of a statement across all DNs. It is counted by ones in a column-store table and by ten thousands in a row-store table.
iops_skew_percent	integer	I/O skew across DNs.
warning	text	Warning. The following warnings and warnings related to SQL self-diagnosis tuning are displayed: 1. Spill file size large than 256MB 2. Broadcast size large than 100MB 3. Early spill 4. Spill times is greater than 3 5. Spill on memory adaptive 6. 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

GS_WLM_SQL_ALLOW displays the configured resource management SQL whitelist, including the default SQL whitelist and the SQL whitelist configured using 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 14-129 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 14-130 GS_WORKLOAD_SQL_ELAPSE_TIME columns

Name	Type	Description
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

Name	Type	Description
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 14-131 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

Name	Type	Description
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 14-132 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 14-133 PG_AVAILABLE_EXTENSION_VERSIONS columns

Name	Type	Description
name	name	Extension name
version	text	Version name
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.

Name	Type	Description
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 14-134 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 14-135 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 14-136 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**.

Name	Type	Description
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 14-137 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. The unit is microsecond.
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 14-138 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 14-139 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

Name	Type	Description
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 14-140 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.
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 14-141 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 14-142 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 14-143 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 (that is, it allows rows to be retrieved in a nonsequential manner). If it is, its value is **true**.
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 14-144 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.
inherited	boolean	-	Includes inherited sub-columns if the value is **true**; otherwise, indicates the column in a specified table.

Name	Type	Reference	Description
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 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. • The number of distinct values is unknown if the value is **0**.
n_dndistinct	real	-	Number of unique not-null data values in the **dn1** column combination • Exact number of distinct values if the value is greater than 0 • Negative of the number of distinct values divided by the number of rows if the value is less than 0 For example, if a value in a column combination appears twice in average, **n_dndistinct** equals **-0.5**. • The number of distinct values is unknown if the value is **0**.

Name	Type	Reference	Description
most_commo n_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**.
most_commo n_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_commo n_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_commo n_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 14-145 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

Name	Type	Description
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 14-146 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 14-147 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 14-148 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)

14.3.102 PG_JOB

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 14-149 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 job has not yet 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 If a job fails to be executed for 16 consecutive times, **job_status** is automatically set to **d**, and no more attempt will be made on this job. **NOTE** ● 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 times the job has started and failed. If a job fails to be executed for 16 consecutive times, no more attempt will be made on it.

Name	Type	Description
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
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 14-150 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 14-151 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 job has not yet 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 If a job fails to be executed for 16 consecutive times, **job_status** is automatically set to **d**, and no more attempt will be made on this job. **NOTE** ● 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 times the job has started and failed. If a job fails to be executed for 16 consecutive times, no more attempt will be made on it.
interval	text	Job execution interval

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

14.3.105 PG_LIFECYCLE_DATA_DISTRIBUTE

PG_LIFECYCLE_DATA_DISTRIBUTE displays the distribution of cold and hot data in a multi-temperature table of OBS.

Table 14-152 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
switchable partition	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

Name	Type	Description
switchable datasize	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 14-153 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**.

Name	Type	Reference	Description
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**.
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 14-154 PG_NODE_ENV columns

Name	Type	Description
node_name	text	Name of the current node
host	text	Host name of the current node
process	integer	Process ID of the current node
port	integer	Port ID of the current node
installpath	text	Installation directory of current node

Name	Type	Description
datapath	text	Data directory of the current node
log_directory	text	Log directory of the current node

14.3.108 PG_OS_THREADS

PG_OS_THREADS displays the status information about all the threads under the current node.

Table 14-155 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 14-156 PG_POOLER_STATUS columns

Name	Type	Description
database	text	Database name
user_name	text	User name
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

Name	Type	Description
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.

14.3.110 PG_PREPARED_STATEMENTS

PG_PREPARED_STATEMENTS displays all prepared statements that are available in the current session.

Table 14-157 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 14-158 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 14-159 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 14-160 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 14-161 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	-	Sets the maximum number of concurrent connections this role can make if this role can log in. -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

Name	Type	Reference	Description
roluseft	boolean	**PG_AUTHID**.roluseft	Whether the role can perform operations on foreign tables
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 14-162 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 the running transaction information on the current node.

Table 14-163 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; **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

Name	Type	Description
prepare_xid	xid	Transaction ID in the **prepared** status. 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

14.3.117 PG_SECLABELS

PG_SECLABELS displays information about security labels.

Table 14-164 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 14-165 PG_SESSION_WLMSTAT columns

Column	Type	Description
datid	oid	OID of the database this backend is connected to
datname	name	Name of the database the backend is connected to
threadid	bigint	ID of the backend thread
processid	integer	Thread PID of the backend
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. This column is reserved.
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 14-166 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 14-167 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, 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**.

The name stems from the fact that this table should not be readable by the public since it contains passwords. **PG_USER** is a publicly readable view on **PG_SHADOW** that blanks out the password column.

Table 14-168 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 14-169 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 14-170 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 14-171 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 true if the backend is currently waiting for a lock or node.

Name	Type	Description
enqueue	text	Queuing status of a statement. Its value can be: ● **waiting in queue**: The statement is in the queue. ● **waiting in global queue**: The statement is in the global queue. ● **waiting in respool queue**: The statement is in the resource pool queue. ● **waiting in ccn queue**: The job is in the CCN queue. ● 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. For example, after user **judy** is connected to the database, the state information of user **joe** and the initial userdbadmin in **pg_stat_activity** is empty. SELECT datname, usename, usesysid, state,pid FROM pg_stat_activity; datname | usename | usesysid | state | pid ----------+---------+----------+--------+----------------- postgres | dbadmin | 10 | | 139968752121616 postgres | dbadmin | 10 | | 139968903116560 db_tpcds | judy | 16398 | active | 139968391403280 postgres | dbadmin | 10 | | 139968643069712 postgres | dbadmin | 10 | | 139968680818448 postgres | joe | 16390 | | 139968563377936 (6 rows)
resource_pool	name	Resource pool used by the user
stmt_type	text	Statement type
query_id	bigint	ID of a query

Name	Type	Description
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 access informaton about all indexes in the database, with information about each index displayed in a row.

Indexes can be used via either simple index scans or "bitmap" index scans. In a bitmap scan the output of several indexes can be combined via AND or OR rules, so it is difficult to associate individual heap row fetches with specific indexes when a bitmap scan is used. Therefore, a bitmap scan increments the **pg_stat_all_indexes.idx_tup_read** count(s) for the index(es) it uses, and it increments the **pg_stat_all_tables.idx_tup_fetch** count for the table, but it does not affect **pg_stat_all_indexes.idx_tup_fetch**.

Table 14-172 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 access information about all rows in all tables (including TOAST tables) in the database.

Table 14-173 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
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**)

Name	Type	Description
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.

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 14-174 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 14-175 PG_STAT_BGWRITER columns

Name	Type	Description
checkpoints_ti med	bigint	Number of scheduled checkpoints that have been performed
checkpoints_r eq	bigint	Number of requested checkpoints that have been performed
checkpoint_wr ite_time	double precision	Total amount of time that has been spent in the portion of checkpoint processing where files are written to disk, in milliseconds
checkpoint_sy nc_time	double precision	Total amount of time that has been spent in the portion of checkpoint processing where files are synchronized to disk, in milliseconds
buffers_check point	bigint	Number of buffers written during checkpoints
buffers_clean	bigint	Number of buffers written by the background writer
maxwritten_cl ean	bigint	Number of times the background writer stopped a cleaning scan because it had written too many buffers
buffers_backe nd	bigint	Number of buffers written directly by a backend
buffers_backe nd_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 14-176 PG_STAT_DATABASE columns

Name	Type	Description
datid	oid	Database OID
datname	name	Database name

Name	Type	Description
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.
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.

Name	Type	Description
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 14-177 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 14-178 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 14-179 PG_STAT_USER_FUNCTIONS columns

Name	Type	Description
funcid	oid	Function OID
schemaname	name	Schema name
funcname	name	Function name
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 14-180 PG_STAT_USER_INDEXES columns

Name	Type	Description
relid	oid	Table OID for the index
indexrelid	oid	Index OID
schemaname	name	Schema name for the index
relname	name	Table name for the index
indexrelname	name	Index name
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 14-181 PG_STAT_USER_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 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)

Name	Type	Description
n_live_tup	bigint	Estimated number of live rows
n_dead_tup	bigint	Estimated number of dead rows
last_vacuum	timestamp with time zone	Last time at which this table was manually vacuumed (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

14.3.135 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 14-182 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

Name	Type	Description
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)
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 14-183 PG_STAT_SYS_INDEXES columns

Name	Type	Description
relid	oid	Table OID for the index
indexrelid	oid	Index OID
schemaname	name	Schema name for the index
relname	name	Table name for the index
indexrelname	name	Index name
idx_scan	bigint	Number of index scans
idx_tup_read	bigint	Number of index entries returned by scans on this index

Name	Type	Description
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 14-184 PG_STAT_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 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 HOT updated (that is, with no separate index update required)
n_live_tup	bigint	Estimated number of live rows
n_dead_tup	bigint	Estimated number of dead rows
last_vacuum	timestamp with time zone	Last time 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

Name	Type	Description
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 14-185 PG_STAT_XACT_ALL_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
n_tup_del	bigint	Number of rows deleted
n_tup_hot_upd	bigint	Number of rows HOT updated (that is, with no separate index update required)

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 14-186 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
n_tup_del	bigint	Number of rows deleted
n_tup_hot_upd	bigint	Number of rows with HOT updates (that is, without separate index required for updates)

14.3.140 PG_STAT_XACT_USER_FUNCTIONS

PG_STAT_XACT_USER_FUNCTIONS displays statistics about function executions, with statistics about each execution displayed in a row.

Table 14-187 PG_STAT_XACT_USER_FUNCTIONS columns

Name	Type	Description
funcid	oid	Function OID
schemaname	name	Schema name
funcname	name	Function name
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

Name	Type	Description
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 14-188 PG_STAT_XACT_USER_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
n_tup_del	bigint	Number of rows deleted
n_tup_hot_upd	bigint	Number of rows HOT updated (that is, with no separate index update required)

14.3.142 PG_STATIO_ALL_INDEXES

PG_STATIO_ALL_INDEXES contains each row of each index in the current database, showing I/O statistics about accesses to that specific index.

Table 14-189 PG_STATIO_ALL_INDEXES columns

Name	Type	Description
relid	oid	Table OID for the index
indexrelid	oid	Index OID

Name	Type	Description
schemaname	name	Schema name for the index
relname	name	Table name for the index
indexrelname	name	Index name
idx_blks_read	bigint	Number of disk blocks read from this index
idx_blks_hit	bigint	Number of buffer hits in this index

14.3.143 PG_STATIO_ALL_SEQUENCES

PG_STATIO_ALL_SEQUENCES contains each row of each sequence in the current database, showing I/O statistics about accesses to that specific sequence.

Table 14-190 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 the sequence
blks_read	bigint	Number of disk blocks read from this sequence
blks_hit	bigint	Number of buffer hits in this sequence

14.3.144 PG_STATIO_ALL_TABLES

PG_STATIO_ALL_TABLES contains one row for each table in the current database (including TOAST tables), showing I/O statistics about accesses to that specific table.

Table 14-191 PG_STATIO_ALL_TABLES columns

Name	Type	Description
relid	oid	Table OID
schemaname	name	Schema name of the table
relname	name	Table name
heap_blks_read	bigint	Number of disk blocks read from this table
heap_blks_hit	bigint	Number of buffer hits in this table

Name	Type	Description
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 14-192 PG_STATIO_SYS_INDEXES columns

Name	Type	Description
relid	oid	Table OID for the index
indexrelid	oid	Index OID
schemaname	name	Schema name for the index
relname	name	Table name for the index
indexrelname	name	Index name
idx_blks_read	bigint	Number of disk blocks read from this 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 14-193 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 the sequence
blks_read	bigint	Number of disk blocks read from this 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 14-194 PG_STATIO_SYS_TABLES columns

Name	Type	Description
relid	oid	Table OID
schemaname	name	Schema name of the table
relname	name	Table name
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 all indexes 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 14-195 PG_STATIO_USER_INDEXES columns

Name	Type	Description
relid	oid	Table OID for the index
indexrelid	oid	Index OID
schemaname	name	Schema name for the index
relname	name	Table name for the index
indexrelname	name	Index name
idx_blks_read	bigint	Number of disk blocks read from this 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 14-196 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 this sequence
blks_hit	bigint	Number of cache hits in this 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 14-197 PG_STATIO_USER_TABLES columns

Name	Type	Description
relid	oid	Table OID
schemaname	name	Schema name of the table
relname	name	Table name
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

14.3.151 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 14-198 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

Name	Type	Description
smpid	integer	Concurrent thread ID
wait_status	text	Waiting status of the current thread. For details about the waiting status, see **Table 14-199**.
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 14-199 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
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

Value	Description
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
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

Value	Description
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.
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.

Value	Description
HashJoin - [build hash | write file]	The **HashJoin** operator is being executed. In this phase, you need to pay attention to the execution time-consuming. ● **build hash**: The **HashJoin** operator is creating a hash table. ● **write file**: The **HashJoin** operator is writing data to disks.
HashAgg - [build hash | write file]	The **HashAgg** operator is being executed. In this phase, you need to pay attention to the execution time-consuming. ● **build hash**: The **HashAgg** operator is creating a hash table. ● **write file**: The **HashAgg** operator is writing data to disks.
HashSetop - [build hash | write file]	The **HashSetop** operator is being executed. In this phase, you need to pay attention to the execution time-consuming. ● **build hash**: The **HashSetop** operator is creating a hash table. ● **write file**: The **HashSetop** operator is writing data to disks.
Sort | Sort - write file	The **Sort** operator is being executed. **write file** indicates that the **Sort** operator is writing data to disks.
Material | Material - write file	The **Material** operator is being executed. **write file** indicates that the **Material** operator is writing data to disks.
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 14-200 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
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.

wait_event	Description
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
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

wait_event	Description
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
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

wait_event	Description
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*.
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.

wait_event	Description
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 14-201 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.

wait_event	Description
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.
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.

wait_event	Description
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.
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.

wait_event	Description
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 14-202 List of wait events corresponding to transaction locks

wait_event	Description
relation	Adds a lock to a table.
extend	Adds a lock to a table being scaled out.
partition	Adds a lock to a partitioned table.
partition_seq	Adds a lock to a partition of a partitioned table.
page	Adds a lock to a table page.
tuple	Adds a lock to a tuple on a page.
transactionid	Adds a lock to a transaction ID.
virtualxid	Adds a lock to a virtual transaction ID.
object	Adds a lock to an object.
cstore_freespace	Adds a lock to idle column-store space.
userlock	Adds a lock to a user.
advisory	Adds an advisory lock.

14.3.152 PG_TABLES

PG_TABLES displays access to each table in the database.

Table 14-203 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.
last_ddl_time	timestamp with time zone	**PG_OBJECT**.mtime	Last time when the cluster was modified.

14.3.153 PG_TDE_INFO

PG_TDE_INFO displays the encryption information about the current cluster.

Table 14-204 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 14-205 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 14-206 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 14-207 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 14-208 PG_TOTAL_USER_RESOURCE_INFO columns

Name	Type	Description
username	name	Username
used_memory	integer	Used memory (unit: MB)
total_memory	integer	Available memory (unit: MB). **0** indicates that the available memory is not limited and depends on the maximum memory available in the database.
used_cpu	double precision	Number of CPU cores in use. 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 storage space (unit: KB). **-1** indicates that the storage space is not limited.
used_temp_space	bigint	Used temporary table storage space (unit: KB)
total_temp_space	bigint	Available temporary table storage space (unit: KB). **-1** indicates that the storage space is not limited.
used_spill_space	bigint	Size of the used operator flushing space, in KB
total_spill_space	bigint	Size of the available operator flushing space, in KB. The value **-1** indicates that the operator flushing space is not limited.
read_kbytes	bigint	CN: total number of bytes read by a user's complex jobs on all DNs in the last 5 seconds. The unit is KB. DN: total number of bytes read by a user's complex jobs from the instance startup time to the current time. The unit is KB.

Name	Type	Description
write_kbytes	bigint	CN: total number of bytes written by a user's complex jobs on all DNs in the last 5 seconds. The unit is KB. DN: 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. Unit: count. DN: total number of read times of a user's complex jobs from the instance startup time to the current time. Unit: count.
write_counts	bigint	CN: total number of write times of a user's complex jobs on all DNs in the last 5 seconds. Unit: count. DN: total number of write times of a user's complex jobs from the instance startup time to the current time. Unit: count.
read_speed	double precision	CN: average read rate of a user's complex jobs on a single DN in the last 5 seconds. (Unit: KB/s) DN: indicates the average read rate of a user's complex jobs on a single DN in the last 5 seconds. (Unit: KB/s)
write_speed	double precision	CN: average write rate of a user's complex jobs on a single DN in the last 5 seconds. (Unit: KB/s) DN: average write rate of a user's complex jobs on a single DN in the last 5 seconds. (Unit: KB/s)

14.3.159 PG_USER

PG_USER displays information about users who can access the database.

Table 14-209 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

Name	Type	Description
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
parentid	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 14-210 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 14-211 PG_VIEWS columns

Name	Type	Reference	Description
schemaname	name	**PG_NAMESPACE**.nspname	Name of the schema that contains the view
viewname	name	**PG_CLASS**.relname	View name
viewowner	name	**PG_AUTHID**.Erolname	Owner of the view
definition	text	-	Definition of the view

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 14-212 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.
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

Name	Type	Description
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 14-213 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 14-214 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 **GS_COLUMN_TABLE_IO_STAT**.

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 14-215 PGXC_COMM_CLIENT_INFO columns

Name	Type	Description
node_name	text	Current node name.

Name	Type	Description
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.

14.3.167 PGXC_COMM_DELAY

PGXC_COMM_STATUS displays the communication library delay status for all the DNs.

Table 14-216 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. Its 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 14-217 PGXC_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.

Name	Type	Description
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 14-218 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

Name	Type	Description
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 14-219 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.

Name	Type	Description
%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 14-220 PGXC_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

14.3.172 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 14-221 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.
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

Name	Type	Description
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 14-222 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 14-223 PGXC_GET_STAT_ALL_PARTITIONS columns

Name	Type	Description
relid	oid	Table OID
partid	oid	Partition OID
schename	name	Schema name of a table
relname	name	Table name
partname	name	Partition name
n_tup_ins	numeric	Number of inserted tuples

Name	Type	Description
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 14-224 PGXC_GET_TABLE_SKEWNESS columns

Name	Type	Description
schemaname	name	Schema name of a table
tablename	name	Name of a table

Name	Type	Description
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(4,3)	Ratio of the maximum table size on a single DN to the total table size
minratio	numeric(4,3)	Ratio of the minimum table size on a single DN to the total table size
skewsize	bigint	Table skew rate (the maximum table size on a single DN minus the minimum table size on a single DN)
skewratio	numeric(4,3)	Table skew rate (skew size divided by total table size)
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 14-225 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
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.

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 14-226 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.

Name	Type	Description
lock_wait_hierarchy	text	Lock wait hierarchy , in the format of *Node name: Process ID->Waiting process ID->Waiting process ID->...*
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

Name	Type	Description
query_start	timestamp with time zone	Start time of the active query
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. For details about the field, see **GS_INSTR_UNIQUE_SQL**.

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 14-227 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.
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

Name	Type	Description
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 14-228 PGXC_NODE_ENV columns

Name	Type	Description
node_name	text	Names of all nodes in the cluster
host	text	Host names of all nodes in the cluster
process	integer	Process IDs of all nodes in the cluster
port	integer	Port numbers of all nodes in the cluster
installpath	text	Installation directory of all nodes in the cluster
datapath	text	Data directory of all nodes in the cluster
log_directory	text	Log directory of all 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.

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.

14.3.184 PGXC_OS_THREADS

PGXC_OS_THREADS displays thread status information under all normal nodes in the current cluster.

Table 14-229 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 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 14-230 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.

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.

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.

14.3.189 PGXC_RESPOOL_RUNTIME_INFO

PGXC_RESPOOL_RUNTIME_INFO displays the running information about all resource pool jobs on all CNs.

Table 14-231 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 14-232 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

Name	Type	Description
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 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.
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.

14.3.191 PGXC_RESPOOL_RESOURCE_HISTORY

PGXC_RESPOOL_RESOURCE_HISTORY is used to query historical monitoring information about resource pools on all instances.

Table 14-233 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

Name	Type	Description
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, 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 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 **GS_ROW_TABLE_IO_STAT**.

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 14-234 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.

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 14-235 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	Thread PID of the backend
usesysid	oid	OID of the user who logged into the backend

Name	Type	Description
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. This column is reserved.
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**: unknown status

Name	Type	Description
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

14.3.196 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 14-236 PGXC_STAT_ACTIVITY columns

Name	Type	Description
coorname	text	Name of the CN in the current cluster

Name	Type	Description
datid	oid	OID of the database that the user session connects to in the backend
datname	name	Name of the database that the user session connects to in the backend
pid	bigint	ID of the backend thread
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.
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

Name	Type	Description
waiting	boolean	The value is true if the backend is currently waiting for a lock or node.
enqueue	text	Queuing status of a statement. Its value can be: ● **waiting in global queue**: The statement is in the global queue. ● **waiting in respool queue**: The statement is in the resource pool queue. ● **waiting in ccn queue**: The job is in the CCN queue. ● 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. For example, after user **judy** is connected to the database, the state information of user **joe** and the initial user **dbadmin** in **pgxc_stat_activity** is empty.
		<pre>SELECT datname, usename, usesysid, state,pid FROM pgxc_stat_activity;
		 datname | usename | usesysid | state | pid
		----------+---------+----------+--------
		+-----------------
		 gaussdb | dbadmin | 10 | | 139968752121616
		 gaussdb | dbadmin | 10 | | 139968903116560
		 db_tpcds | judy | 16398 | active | 139968391403280
		 gaussdb | dbadmin | 10 | | 139968643069712
		 gaussdb | dbadmin | 10 | | 139968680818448
		 gaussdb | joe | 16390 | | 139968563377936
		(6 rows)</pre>

resource_pool	name	Resource pool used by the user
stmt_type	text	Type of a user statement
query_id	bigint	ID of a query

Name	Type	Description
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 14-237 PGXC_STAT_BAD_BLOCK columns

Name	Type	Description
nodename	text	Node name
databaseid	integer	Database OID

Name	Type	Description
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.

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.

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. This view is accessible only to users with system administrators rights.

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 14-238 PGXC_STAT_TABLE_DIRTY columns

Name	Type	Description
nodename	text	Node name

Name	Type	Description
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**
last_analyze	timestampwith time zone	Time of the last manual **ANALYZE**
last_antoanalyze	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 with HOT updates
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 14-115**.

📖 **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. For details about the columns, see **GS_TABLE_CHANGE_STAT**.

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. For details about the columns, see **GS_TABLE_STAT**.

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 14-239 PGXC_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 14-199**.
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 14-240 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 14-241 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 14-242 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 14-243 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 14-244 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 14-245 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.

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 by Database Manager to query data from a database. 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 14-4**.

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 14-4**.

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. For details about columns in the view, see **Table 14-126**.

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**.

For details about columns in the view, see **Table 14-127**.

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**.

For details about columns in the view, see **Table 14-127**.

14.3.218 PGXC_WLM_SESSION_STATISTICS

PGXC_WLM_SESSION_STATISTICS displays load management information about jobs that are being executed on CNs.

For details about columns in the view, see **Table 14-128**.

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 14-246 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
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 14-247 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
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 14-248 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

Name	Type	Description
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

14.3.222 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 14-249 PGXC_WORKLOAD_TRANSACTION columns

Name	Type	Description
node_name	name	Node name

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 (unit: μs)
resp_max	bigint	Maximum response time (unit: μs)
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 14-250 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** ordinary 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 PLAN_TABLE_DATA

PLAN_TABLE_DATA displays 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 14-251 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.

14.3.225 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 14-252 PV_FILE_STAT columns

Name	Type	Description
filenum	oid	File ID
dbid	oid	Database ID
spcid	oid	Tablespace ID
phyrds	bigint	Number of times of reading physical files
phywrts	bigint	Number of times of writing into physical files
phyblkrd	bigint	Number of times of reading physical file blocks
phyblkwrt	bigint	Number of times of writing into physical file blocks
readtim	bigint	Total duration of reading files. The unit is microsecond.
writetim	bigint	Total duration of writing files. The unit is microsecond.
avgiotim	bigint	Average duration of reading and writing files. The unit is microsecond.
lstiotim	bigint	Duration of the last file reading. The unit is microsecond.
miniotim	bigint	Minimum duration of reading and writing files. The unit is microsecond.

Name	Type	Description
maxiowtm	bigint	Maximum duration of reading and writing files. The unit is microsecond.

14.3.226 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 14-253 PV_INSTANCE_TIME columns

Name	Type	Description
stat_id	integer	Type ID
stat_name	text	Running time type name
value	bigint	Running time value

14.3.227 PV_OS_RUN_INFO

PV_OS_RUN_INFO displays the running status of the current operating system.

Table 14-254 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

Name	Type	Description
comments	text	Remarks of the OS running status
cumulative	boolean	Whether the value of the OS running status is cumulative

14.3.228 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 14-255 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
used_mem	integer	Memory allocated to the currently executed task, in MB
peak_mem	integer	Peak memory allocated to the currently executed task, in MB

14.3.229 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 14-256 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
```

14.3.230 PV_SESSION_STAT

PV_SESSION_STAT displays session state statistics based on session threads or the **AutoVacuum** thread.

Table 14-257 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

14.3.231 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 14-258 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

14.3.232 PV_TOTAL_MEMORY_DETAIL

PV_TOTAL_MEMORY_DETAIL displays statistics about memory usage of the current database node in the unit of MB.

Table 14-259 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

14.3.233 PV_REDO_STAT

PV_REDO_STAT displays statistics on redoing Xlogs on the current node.

Table 14-260 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

14.3.234 REDACTION_COLUMNS

REDACTION_COLUMNS displays information about all redaction columns in the current database.

Table 14-261 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.

14.3.235 REDACTION_POLICIES

REDACTION_POLICIES displays information about all redaction objects in the current database.

Table 14-262 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.

14.3.236 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. For details about the columns, see **GS_TABLE_STAT**.

14.3.237 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

14.3.238 USER_CONSTRAINTS

USER_CONSTRAINTS displays the table constraint information accessible to the current user.

Name	Type	Description
constraint_name	vcharacter varying(64)	Constraint name

Name	Type	Description
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)

14.3.239 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

14.3.240 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

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

14.3.241 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

14.3.242 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

Name	Type	Description
column_expression	text	The function-based index expression of a specified column
column_position	smallint	Position of column in the index

14.3.243 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.

Name	Type	Description
def_tablespac e_name	name	Tablespace name of the index partition

14.3.244 USER_JOBS

USER_JOBS displays all jobs owned by the user.

Table 14-263 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.

Name	Type	Description
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
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 times the job has started and failed. If a job fails to be executed for 16 consecutive times, no more attempt will be made on it.
what	text	Body of the PL/SQL blocks or anonymous clock that the job executes

14.3.245 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**.

14.3.246 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
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

14.3.247 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

Name	Type	Description
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
partitioning_key_count	integer	Number of partition keys of the partitioned table

14.3.248 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

14.3.249 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

14.3.250 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

14.3.251 USER_SYNONYMS

USER_SYNONYMS displays synonyms accessible to the current user.

Table 14-264 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.

14.3.252 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

Name	Type	Description
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
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**.

14.3.253 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 the table or the view
table_name	character varying(64)	Name of the table or the view

Name	Type	Description
comments	text	Comments

14.3.254 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

14.3.255 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
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	The estimated number of rows in the table

14.3.256 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

14.3.257 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

14.3.258 V$SESSION

V$SESSION displays all session information about the current session.

Table 14-265 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

14.3.259 V$SESSION_LONGOPS

V$SESSION_LONGOPS displays the progress of ongoing operations.

Table 14-266 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 Collation Support

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.

16 GUC Parameters

16.1 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.

- You can also connect to a cluster and run SQL commands to check the GUC parameters.

  - Run the **SHOW** command.

    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;

16.2 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. This type of parameters take effect immediately after they are set. You do not need to restart the cluster. 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. This type of parameters take effect immediately after they are set. You do not need to restart the cluster. 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. This type of parameters take effect only after the cluster is restarted. After you modify a parameter of this type, the system displays a message indicating that the cluster is to be restarted. You are advised to manually restart the cluster during off-peak hours for the setting to take effect.
  - SIGHUP: global database parameters. This type of parameters take effect globally and cannot take effect for single sessions.
  - BACKEND: global database parameters. This type of parameters take effect globally and cannot take effect for single 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

16.3 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.

Precautions

- If the value range of a parameter is a string, the string should comply with the naming conventions of the path and file name in the OS running the 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.

16.4 Connection and Authentication

16.4.1 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 8.1.3 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.

16.4.2 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**

auth_iteration_count

Parameter description: Specifies the number of interactions during the generation of encryption information for authentication.

Type: SIGHUP

Value range: an integer ranging from 2048 to 134217728

Default value: **10000**

NOTICE

If this parameter is set to a large value, performance deteriorates in operations involving password encryption, such as authentication and user creation. Set this parameter to an appropriate value based on the hardware conditions.

---

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

Parameter description: Specifies whether the SSL connection is enabled.

Type: POSTMASTER

Value range: Boolean

- **on** indicates that the SSL connection is enabled.
- **off** indicates that the SSL connection is not enabled.

NOTICE

GaussDB(DWS) supports the SSL connection when the client connects to CNs. It is recommended that the SSL connection be enabled only on CNs.

Default value: **on**

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**

- **0** indicates that passwords are encrypted in MD5 mode.
- **1** indicates that passwords are encrypted using SHA-256, which is compatible with the MD5 user authentication method of the PostgreSQL client.
- **2** indicates that passwords are encrypted using SHA-256.

NOTICE

- MD5 is not recommended because it is not a secure encryption algorithm.
- If the cluster is upgraded from 8.0.0 or an earlier version to the current version, the default value of this parameter is the same as that of the cluster of the earlier version. For example, the default value of **password_encryption_type** in 8.0.0 is **1**. After the cluster is upgraded from 8.0.0 to 8.1.1, the default value of **password_encryption_type** remains **1**.

Default value: 2

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 16-1 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	/

ID	Chara cter	ID	Charac ter	ID	Charac ter	ID	Charact er
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**

16.4.3 Communication Library Parameters

This section describes parameter settings and value ranges for communication libraries.

comm_tcp_mode

Parameter description: Specifies whether the communication library uses the TCP or SCTP protocol to set up a data channel. The modification of this parameter takes effect after the cluster is restarted.

Type: POSTMASTER

Value range: Boolean. If this parameter is set to **on** for CNs, the CNs connect to DNs using TCP. If this parameter is set to **on** for DNs, the DNs communicate with each other using TCP.

Default value: **on**

comm_sctp_port

Parameter description: Specifies the TCP or SCTP listening port used by the TCP proxy communication library or SCTP communication library, respectively.

Type: POSTMASTER

NOTICE

This port number is automatically allocated during cluster deployment. Do not change the parameter setting. If the port number is incorrectly set, the database communication fails.

Value range: an integer ranging from 0 to 65535

Default value: port + Number of primary DNs on the local host x 2 + Sequence number of the local DN on the local host

comm_control_port

Parameter description: Specifies the TCP listening port used by the TCP proxy communication library or SCTP communication library, respectively.

Type: POSTMASTER

Value range: an integer ranging from 0 to 65535

Default value: port + Number of primary DNs on the local host x 2 + Sequence number of the local DN on the local host + 1

NOTICE

This port number is automatically allocated during cluster deployment. Do not change the parameter setting. If the port number is incorrectly set, the database communication fails.

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**.

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_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

16.5 Resource Consumption

16.5.1 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.

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(**max_process_memory** x 1024/18,2),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(**max_process_memory** x 1024/18,2),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

16.5.2 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 **sql_use_spacelimit** to 10% of the total space of the disk where the instance is.

> **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.

16.5.3 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**

16.5.4 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

Context

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. Adjusting vacuum's resource consumption is best done by changing other parameters.

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**

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

16.5.5 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**

16.6 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**

enable_delta_store

Parameter description: This parameter has been discarded. You can set this parameter to **on** for forward compatibility, but the setting will not take effect.

For details about how to enable the delta table function of column-store tables, see the table-level parameter **enable_delta**.

Type: POSTMASTER

Value range: Boolean

- **on** indicates that the delta table function of column-store tables is enabled.
- **off** indicates that the delta table function of column-store tables is disabled.

Default value: **off**

16.7 Write Ahead Logs

16.7.1 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**

16.7.2 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.

---

checkpoint_completion_target

Parameter description: Specifies the target of checkpoint completion, as a fraction of total time between checkpoints.

Type: SIGHUP

Value range: 0.0 to 1.0. The default value 0.5 indicates that each checkpoint must be completed within 50% of the checkpoint interval.

Default value: **0.5**

checkpoint_warning

Parameter description: Specifies a time in seconds. If the checkpoint interval is close to this time due to filling of checkpoint segment files, a message is sent to the server log to increase the **checkpoint_segments** value.

Type: SIGHUP

Value range: an integer (unit: s). **0** indicates that warning is disabled.

Default value: **5min**

Recommended value: **5min**

checkpoint_wait_timeout

Parameter description: Specifies the longest time that the checkpoint waits for the checkpointer thread to start.

Type: SIGHUP

Value range: an integer ranging from 2 to 3600 (s)

Default value: **1min**

16.7.3 Archiving

archive_mode

Parameter description: When **archive_mode** is enabled, completed WAL segments are sent to archive storage by setting **archive_command**.

Type: SIGHUP

Value range: Boolean

- **on**: The archiving is enabled.
- **off**: The archiving is disabled.

Default value: **off**

NOTICE

When **wal_level** is set to **minimal**, **archive_mode** cannot be used.

---

archive_command

Parameter description: Specifies the command used to archive WALs set by the administrator. You are advised to set the archive log path to an absolute path.

Type: SIGHUP

Value range: a string

Default value: **(disabled)**

> **NOTICE**
>
> - Any **%p** in the string is replaced with the absolute path of the file to archive, and any **%f** is replaced with only the file name. (The relative path is relative to the data directory.) Use **%%** to embed an actual **%** character in the command.
> - This command returns zero only if it succeeds. Example:
>   ```
>   archive_command = 'cp --remove-destination %p /mnt/server/archivedir/%f'
>   archive_command = 'copy %p /mnt/server/archivedir/%f'  # Windows
>   ```
> - **--remove-destination** indicates that files will be overwritten during the archiving.
> - When **archive_mode** is set to **on** or not specified, a **backup** folder will be created in the **pg_xlog** directory and WALs will be compressed and copied to the **pg_xlog/backup** directory.

max_xlog_backup_size

Parameter description: Specifies the size of WAL logs backed up in the **pg_xlog/backup** directory.

Type: SIGHUP

Value range: an integer between **1048576** and **104857600**. The unit is KB.

Default value: 2097152

> **NOTICE**
>
> - The **max_xlog_backup_size** parameter setting takes effect only when **archive_mode** is enabled and **archive_command** is set to **NULL**.
> - The system checks the size of backup WALs in the **pg_xlog/backup** directory every minute. If the size exceeds the value specified by **max_xlog_backup_size**, the system deletes the earliest backup WALs until the size is less than the **max_xlog_backup_size** value × 0.9.

archive_timeout

Parameter description: Specifies the archiving period.

Type: SIGHUP

Value range: an integer ranging from 0 to INT_MAX. The unit is second. **0** indicates that archiving timeout is disabled.

Default value: **0**

> **NOTICE**
>
> ● The server is forced to switch to a new WAL segment file with the period specified by this parameter.
>
> ● Archived files that are closed early due to a forced switch are still of the same length as completely full files. Therefore, a very short **archive_timeout** will bloat the archive storage. You are advised to set **archive_timeout** to **60s**.

16.8 HA Replication

16.8.1 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_replication_slots

Parameter description: Specifies the number of log replication slots on the primary server.

Type: POSTMASTER

Value range: an integer ranging from 0 to 262143

Default value: **8**

A physical replication slot provides an automatic method to ensure that an Xlog is not removed from a primary DN before all the standby and secondary DNs receive

it. Physical replication slots are used to support HA clusters. The number of physical replication slots required by a cluster is as follows: ratio of standby and secondary DNs to the primary DN in a ring of DNs. For example, if an HA cluster has 1 primary DN, 1 standby DN, and 1 secondary DN, the number of required physical replication slots will be 2.

Plan the number of logical replication slots as follows:

- A logical replication slot can carry changes of only one database for decoding. If multiple databases are involved, create multiple logical replication slots.

- If logical replication is needed by multiple target databases, create multiple logical replication slots in the source database. Each logical replication slot corresponds to one logical replication link.

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.

16.8.2 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.

16.9 Query Planning

16.9.1 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**

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

16.9.2 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**

- 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**

16.9.3 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**

16.9.4 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, **default_statistics_target * 300** is used as the size of the random sampling. For example, if the default value is 100, 100 x 300 pages are read in a random sampling.

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 function is enabled. The system dynamically selects the optimal parallelism degree [1,8] (x86 platforms) or [1,64] (Kunpeng platforms) for each query based on the 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.

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**

16.10 Error Reporting and Logging

16.10.1 Logging Destination

log_truncate_on_rotation

Parameter description: Specifies the writing mode of the log files when **logging_collector** is set to **on**.

Type: SIGHUP

Value range: Boolean

- **on** indicates that GaussDB(DWS) overwrites the existing log file of the same name on the server.

- **off** indicates that GaussDB(DWS) appends the log messages to the existing log file of the same name on the server.

Default value: **off**

📖 **NOTE**

Example:

Assume that you plan to keep logs in a period of 7 days, one log file is generated per day, log files generated on Monday are named **server_log.Mon** and named **server_log.Tue** on Tuesday (others are named in the same way), and log files generated on the same day in different weeks are overwritten. Implement the plan by performing the following operations: set **log_filename** to **server_log.%a**, **log_truncate_on_rotation** to **on**, and **log_rotation_age** to **1440** (indicating the valid duration of the log file is 24 hours).

log_rotation_age

Parameter description: Specifies the interval for creating a log file when **logging_collector** is set to **on**. If the difference between the current time and the time when the previous audit log file is created is greater than the value of **log_rotation_age**, a new log file will be generated.

Type: SIGHUP

Value range: an integer ranging from 0 to 24 days. The unit is min, h, or d. **0** indicates that the time-based creation of new log files is disabled.

Default value: **1d**

log_rotation_size

Parameter description: Specifies the maximum size of a server log file when **logging_collector** is set to **on**. If the total size of messages in a server log exceeds the capacity of the server log file, a log file will be generated.

Type: SIGHUP

Value range: an integer ranging from INT_MAX to 1024. The unit is KB.

0 indicates the capacity-based creation of new log files is disabled.

Default value: **20 MB**

event_source

Parameter description: Specifies the identifier of the GaussDB(DWS) error messages in logs when **log_destination** is set to **eventlog**.

Type: POSTMASTER

Value range: a string

Default value: **PostgreSQL**

16.10.2 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 16-2**.

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 16-2**.

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 16-2**.

📖 NOTE

- The default is **error**, indicating that statements causing errors, log messages, fatal errors, or panics will be logged.
- **panic**: This feature is disabled.

Default value: **error**

log_min_duration_statement

Parameter description: Specifies the threshold for logging 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 16-2**.

Default value: **panic**

Table 16-2 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 16-2 Message Severity Levels

Severity	Description	syslog	eventlog
debug[1-5]	Provides detailed debug information.	DEBUG	INFORMATION
log	Reports information of interest to administrators, for example, checkpoint activity.	INFO	INFORMATION
info	Provides information implicitly requested by the user, for example, output from **VACUUM VERBOSE**.	INFO	INFORMATION
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	INFORMATION
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
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. The smaller the value is, the more log data is output, resulting in more negative impact on the 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)
```

16.10.3 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_checkpoints

Parameter description: Specifies whether the statistics on the checkpoints and restart points are recorded in the server logs. When this parameter is set to **on**, statistics on checkpoints and restart points are recorded in the log messages, including the number of buffers to be written and the time spent in writing them.

Type: SIGHUP

Value range: Boolean

- **on** indicates the statistics on the checkpoints and restart points are recorded in the server logs.
- **off** indicates the statistics on the checkpoints and restart points are not recorded in the server logs.

Default value: **off**

log_connections

Parameter description: Specifies whether to record connection request information of the client.

Type: BACKEND

NOTICE

- This is a session connection parameter. You are advised not to configure this parameter.
- Some client programs, such as gsql, attempt to connect twice while determining if a password is required. In this case, duplicate **connection receive** messages do not necessarily indicate a problem.

Value range: Boolean

- **on** indicates the request information is recorded.
- **off** indicates the request information is not recorded.

Default value: **off**

log_disconnections

Parameter description: Specifies whether to record end connection request information of the client.

Type: BACKEND

Value range: Boolean

- **on** indicates the request information is recorded.
- **off** indicates the request information is not recorded.

Default value: **off**

📖 **NOTE**

This is a session connection parameter. You are advised not to configure this parameter.

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_hostname

Parameter description: By default, connection log messages only show the IP address of the connected host. The host name can be recorded when this parameter is set to **on**. It may take some time to parse the host name. Therefore, the database performance may be affected.

Type: SIGHUP

Value range: Boolean

- **on** indicates the host name can be simultaneously recorded.
- **off** indicates the host name cannot be simultaneously recorded.

Default value: **off**

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**

log_timezone

Parameter description: Specifies the time zone used for time stamps written in the server log. Different from **TimeZone**, this parameter takes effect for all sessions in the database.

Type: SIGHUP

Value range: a string

Default value: **PRC**

> 📖 **NOTE**
>
> The value can be changed when **gs_initdb** is used to set system environments.

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,CA
RBONDATA,SNAPSHOT,XACT,HANDLE,CLOG,TQUAL,EC,REMOTE,CN_RETRY,PLSQL,TEXTSEARCH,SEQ,INSTR,
COMM_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,PL
SQL,TEXTSEARCH,SEQ,INSTR,COMM_IPC,COMM_PARAM,CSTORE,JOB,STREAMPOOL,STREAM_CTESCAN),of
f()
(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**

16.11 Alarm Detection

During cluster running, error scenarios can be detected in a timely manner to inform users as soon as possible.

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**

16.12 Statistics During the Database Running

16.12.1 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 loaded using **reload** and the new value is less than the original value, the Unique SQL statistics collected by 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.

16.12.2 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**

16.13 Workload 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_control_group

Parameter description: Specifies whether to enable the Cgroup management function. This parameter must be applied on both CNs and DNs.

Type: SIGHUP

Value range: Boolean

- **on** indicates the Cgroup management function is enabled.
- **off** indicates the Cgroup management function is disabled.

Default value: **on**

📖 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.

enable_backend_control

Parameter description: Specifies whether to control the database permanent thread to the **DefaultBackend** Cgroup. This parameter must be applied on both CNs and DNs.

Type: POSTMASTER

Value range: Boolean

- **on**: Controls the permanent thread to the **DefaultBackend** Cgroup.
- **off**: Does not control the permanent thread to the **DefaultBackend** Cgroup.

Default value: **on**

enable_vacuum_control

Parameter description: Specifies whether to control the database permanent thread autoVacuumWorker to the **Vacuum** Cgroup. This parameter must be applied on both CNs and DNs.

Type: POSTMASTER

Value range: Boolean

- **on**: Controls the database permanent thread autoVacuumWorker to the **Vacuum** Cgroup.
- **off**: Does not control the database permanent thread autoVacuumWorker to the **Vacuum** Cgroup.

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**

enable_verify_active_statements

Parameter description: Specifies whether to enable the background calibration function in static adaptive load scenarios. This parameter must be used on CNs.

Type: SIGHUP

Value range: Boolean

- **on** indicates the background calibration function is enabled.
- **off** indicates the background calibration function is disabled.

Default value: **on**

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. Threshold for dividing simple jobs and complex jobs. A job whose estimated cost is less than the value of this parameter is a simple job, and a job whose estimated cost is larger than or equal to the value of this parameter is a complex job.

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 11-1**) 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 11-1**) will be archived into historical views.

- If the value is greater than **0**, information about statements recorded in the real-time resource monitoring view (see **Table 11-1**), whose execution time exceeds this value will be archived into historical views.

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_bbox_dump

Parameter description: Specifies whether the black box function is enabled. The core files can be generated even through the core dump mechanism is not configured in the system. This parameter must be applied on both CNs and DNs.

Type: SIGHUP

Value range: Boolean

- **on** indicates that the black box function is enabled.
- **off** indicates that the black box function is disabled.

Default value: **off**

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_statistics_memory

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: SIGHUP

Value range: an integer ranging from 5 MB to 50% of **max_process_memory**

Default value: **5 MB**

session_history_memory

Parameter description: Specifies the memory size of a historical query view.

Type: SIGHUP

Value range: an integer ranging from 10 MB to 50% of **max_process_memory**

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.

16.14 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**. For clusters of 8.1.2 or earlier, configure GUC parameters by following the instructions provided in this section.

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**

log_autovacuum_min_duration

Parameter description: Records each step performed by the automatic cleanup process to the server log when the execution time of the automatic cleanup process is greater than or equal to a certain value. This parameter helps track the automatic cleanup behaviors.

Type: SIGHUP

For example, set the **log_autovacuum_min_duration** parameter to 250 ms to record the information related to the automatic cleanup commands running the parameters whose values are greater than or equal to 250 ms.

Value range: an integer ranging from –1 to INT_MAX. The unit is ms.

- If this parameter is set to **0**, all the automatic cleanup operations are recorded in the log.
- If this parameter is set to **–1**, all the automatic cleanup operations are not recorded in the log.
- If this parameter is not set to **–1**, an automatic cleanup operation is skipped and a message is recorded due to lock conflicts.

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**).
- If **autovacuum_max_workers > 0** and **autovacuum = on** are configured, all tables are cleared.

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**

colvacuum_threshold_scale_factor

Parameter description: Specifies the minimum percentage of dead tuples for vacuum rewriting in column-store tables. A file is rewritten only when the ratio of dead tuples to (all_tuple - null_tuple) in the file is greater than the value of this parameter.

Type: SIGHUP

Value range: an integer ranging from -2 to 100.

- **-2** indicates that vacuum rewriting and vacuum cleanup are not performed.
- **-1** indicates to perform vacuum rewriting is not performed and only vacuum cleanup is performed.
- The value ranges from 0 to 100, indicating the percentage of dead tuples.

Default value: **-2**

16.15 Default Settings of Client Connection

16.15.1 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.

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**

max_compile_functions

Parameter description: Specifies the maximum number of function compilation results stored in the server. Excessive functions and compilation results generated during the storage may occupy large memory space. Setting this parameter to a proper value can reduce the memory usage and improve system performance.

Type: POSTMASTER

Value range: an integer ranging from 1 to INT_MAX

Default value: **1000**

gin_pending_list_limit

Parameter description: Specifies the maximum size of the GIN pending list which is used when **fastupdate** is enabled. If the list grows larger than this maximum size, it is cleaned up by moving the entries in it to the main GIN data structure in batches. This setting can be overridden for individual GIN indexes by modifying index storage parameters.

Type: USERSET

Value range: an integer ranging from 64 to INT_MAX. The unit is KB.

Default value: **4 MB**

16.15.2 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**

16.15.3 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**

16.16 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).

This parameter has a higher priority than **lockwait_timeout** and takes effect only for **AccessExclusiveLock**.

- 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**

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

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

enable_online_ddl_waitlock

Parameter description: Specifies whether to block DDL operations to wait for the release of cluster locks, such as pg_advisory_lock and pgxc_lock_for_backup. This parameter is mainly used in online OM operations and you are not advised to modify the settings.

Type: SIGHUP

Value range: Boolean

- **on** indicates that DDL operations will be blocked to wait for the release of cluster locks.
- **off** indicates that DDL operations will not be blocked.

Default value: **off**

16.17 Version and Platform Compatibility

16.17.1 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**

16.17.2 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**

16.18 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: If this parameter is set to **on** and the client character set of the database is encoded in UTF-8 format, the occurring character encoding conversion errors will be recorded in logs. Additionally, converted characters that have conversion errors will be ignored and replaced 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**

cn_send_buffer_size

Parameter description: Specifies the size of the data buffer used for data transmission on the CN.

Type: POSTMASTER

Value range: an integer ranging from 8 to 128. The unit is KB.

Default value: **8 KB**

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

data_sync_retry

Parameter description: Specifies whether to keep running the database when updated data fails to be written into disks by using the **fsync** function. In some OSs, no error is reported even if **fsync** has failed for multiple times. As a result, data is lost.

Type: POSTMASTER

Value range: Boolean

- **on**: The database keeps running and **fsync** is executed again after **fsync** fails.
- **off**: **PANIC** is reported and the database is stopped after **fsync** fails.

Default value: **off**

16.19 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**

max_coordinators

Parameter description: Specifies the maximum number of CNs in a cluster.

Type: POSTMASTER

Value range: an integer ranging from 2 to 40

Default value: **40**

max_datanodes

Parameter description: Specifies the maximum number of DNs in a cluster.

Type: POSTMASTER

Value range: an integer ranging from 2 to 65535

Default value: 4096

cache_connection

Parameter description: Specifies whether to reclaim the connections of a connection pool.

Type: SIGHUP

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: BACKEND

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**

16.20 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.

Type: SUSET

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**

enable_gtm_free

Parameter description: Specifies whether the GTM-FREE mode is enabled. In large concurrency scenarios, the snapshots delivered by the GTM increase in number and size. The network between the GTM and the CN becomes the performance bottleneck. The GTM-FREE mode is used to eliminate the bottleneck. In this mode, the CN communicates with DNs instead of the GTM. The CN sends queries to each DN, which locally generates snapshots and xids, ensuring external write consistency but not external read consistency.

You are not advised to set this parameter to **on** in OLTP or OLAP scenarios where strong read consistency is required. This parameter is invalid for GaussDB(DWS).

Type: SUSET

Value range: Boolean

- **on** indicates that the GTM-FREE mode is enabled and the cluster ensures eventual read consistency.
- **off** indicates that the GTM-FREE mode is disabled.

Default value: **off**

16.21 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_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**

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**

default_orientation

Parameter description: Specifies the type of the table to be created if no storage method is specified during table creation. The value for each node must be the same. This parameter is supported by version 8.1.3 or later clusters.

Type: SUSET

Value range: **row**, **column**, **column enabledelta**

- **row**: creates a row-store table.
- **column**: creates a column-store table.
- **column enabledelta**: creates a column-store table with delta tables enabled.

Default value: **row**

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_recovery_messages

Parameter description: Specifies whether to enable logging of recovery-related debugging output. This parameter allows users to overwrite the normal setting of **log_min_messages**, but only for specific messages. This is intended for use in debugging the standby server.

Type: SIGHUP

Value range: enumerated values. Valid values include **debug5**, **debug4**, **debug3**, **debug2**, **debug1**, and **log**. For details about the parameter values, see **log_min_messages**.

Default value: **log**

📖 **NOTE**

- **log** indicates that recovery-related debugging information will not be logged.

- Except the default value **log**, each of the other values indicates that recovery-related debugging information at the specified level will also be logged. Common settings of **log_min_messages** will unconditionally record information into server logs.

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**

string_hash_compatible

Parameter description: Specifies whether to use the same method to calculate char-type hash values and varchar- or text-type hash values. Based on the setting of this parameter, you can determine whether a redistribution is required when a distribution column is converted from a char-type data distribution into a varchar- or text-type data distribution.

Type: POSTMASTER

Value range: Boolean

- **on** indicates that the same calculation method is used and a redistribution is not required.
- **off** indicates that different calculation methods are used and a redistribution is required.

📖 NOTE

> Calculation methods differ in the length of input strings used for calculating hash values. (For a char-type hash value, spaces following a string are not counted as the length. For a text- or varchar-type hash value, the spaces are counted.) The hash value affects the calculation result of queries. To avoid query errors, do not modify this parameter during database running once it is set.

Default value: **off**

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_debug_vacuum

Parameter description: Specifies whether to allow output of some VACUUM-related logs for problem locating. This parameter is used only by developers. Common users are advised not to use it.

Type: SIGHUP

Value range: Boolean

- **on/true** indicates that output of vacuum-related logs is allowed.
- **off/false** indicates that output of vacuum-related logs is disallowed.

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**: 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. The number of buckets affects the precision of distinct values calculated by HLL. The more buckets there are, the smaller the deviation is. The deviation range is as follows: [$-1.04/2^{log2m*1/2}$, $+1.04/2^{log2m*1/2}$]

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 16-3**.

Type: USERSET

Value range: an integer ranging from 1 to 5

Default value: **5**

Table 16-3 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 x 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**

pljava_vmoptions

Parameter description: Specifies the startup parameters for JVMs used by the PL/Java function.

Type: SUSET

Value range: a string, supporting:

- JDK8 JVM startup parameters.
- JDK8 JVM system attributes (starting with **–D**, for example, **–Djava.ext.dirs**).
- User-defined parameters (starting with **–D**, for example, **–Duser.defined.option**).

NOTICE

If **pljava_vmoptions** is set to a value beyond the value range, an error will be reported when PL/Java functions are used.

Default value: empty

javaudf_disable_feature

Parameter description: Specifies the granularity of Java UDF actions.

Type: SIGHUP

Value range: a string

- **none** indicates that any action specified in other fine-grained parameters is enabled. When this parameter is set together with other parameters, **none** is invalid.
- **all** indicates that all Java UDF functions are disabled. This option has the highest priority.
- **extdir** indicates that the function of storing dependency JAR packages in a third-party path is disabled.
- **hadoop** indicates that Hadoop functions are disabled.
- **reflection** indicates that the reflection permission (**ReflectPermission**) is disabled during the execution of Java UDF functions.
- **loadlibrary** indicates that the dynamic library loading permission (**loadLibrary**) is disabled during the execution of Java UDF functions.
- **net** indicates that the network permission (**NetPermission**) is disabled during the execution of Java UDF functions.
- **socket** indicates that the socket permission (**SocketPermission**) is disabled during the execution of Java UDF functions.
- **security** indicates that the security configuration permission (**SecurityPermission**) is disabled during the execution of Java UDF functions.
- **classloader** indicates that the **classLoader** creation permission (**createClassLoader**) is disabled during the execution of Java UDF functions.

- **access_declared_members** indicates that the permission of accessing other declared members (**accessDeclaredMembers**) is disabled during the execution of Java UDF functions.

  Default value: **extdir,hadoop,reflection,loadlibrary,net,socket,security,classloader,access_declared_members**

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**

enable_immediate_interrupt

Parameter description: Specifies whether the execution of the current statement or session can be immediately interrupted in the signal processing function.

Type: SIGHUP

Value range: Boolean

- **on** indicates that the execution of the current statement or session can be immediately interrupted in the signal processing function.
- **off** indicates that the execution of the current statement or session cannot be immediately interrupted in the signal processing function.

Default value: **off**

NOTE

Exercise caution when setting this parameter to **on**. If the execution of the current statement or session can be immediately interrupted in the signal processing function, the execution of some key processes may be interrupted, causing the failure to release the global lock in the system. It is recommended that this parameter be set to **on** only during system debugging or fault prevention.

16.22 Auditing

16.22.1 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_data_format

Parameter description: Specifies the format of the audit log files. Currently, only the binary format is supported.

Type: POSTMASTER

Value range: a string

Default value: **binary**

audit_rotation_interval

Parameter description: Specifies the interval of creating an audit log file. If the difference between the current time and the time when the previous audit log file is created is greater than the value of **audit_rotation_interval**, a new audit log file will be generated.

Type: SIGHUP

Value range: an integer ranging from 1 to **INT_MAX/60**. The unit is min.

Default value: **1d**

NOTICE

Adjust this parameter only when required. Otherwise, **audit_resource_policy** may fail to take effect. To control the storage space and time of audit logs, set the **audit_resource_policy**, **audit_space_limit**, and **audit_file_remain_time** parameters.

---

audit_rotation_size

Parameter description: Specifies the maximum capacity of an audit log file. If the total number of messages in an audit log exceeds the value of **audit_rotation_size**, the server will generate a new audit log file.

Type: SIGHUP

Value range: an integer ranging from 1 to 1024. The unit is MB.

Default value: **10 MB**

NOTICE

Adjust this parameter only when required. Otherwise, **audit_resource_policy** may fail to take effect. To control the storage space and time of audit logs, set the **audit_resource_policy**, **audit_space_limit**, and **audit_file_remain_time** parameters.

---

audit_resource_policy

Parameter description: Specifies the policy for determining whether audit logs are preferentially stored by space or time.

Type: SIGHUP

Value range: Boolean

- **on** indicates that audit logs are preferentially stored by space. A maximum of **audit_space_limit** logs can be stored.
- **off** indicates that audit logs are preferentially stored by time. A minimum duration of **audit_file_remain_time** logs must be stored. If the value of **audit_file_remain_time** is too large, the disk space occupied by stored audit logs reaches the value of **audit_space_limit**. In this case, the earliest audit files are deleted.

Default value: **on**

audit_file_remain_time

Parameter description: Specifies the minimum duration required for recording audit logs. This parameter is valid only when **audit_resource_policy** is set to **off**.

Type: SIGHUP

Value range: an integer ranging from 0 to 730. The unit is day. **0** indicates that the storage duration is not limited.

Default value: **90**

audit_space_limit

Parameter description: Specifies the total disk space occupied by audit files.

Type: SIGHUP

Value range: an integer ranging from **1024 KB** to **1024 GB**. The unit is KB.

Default value: **1GB**

audit_file_remain_threshold

Parameter description: Specifies the maximum number of audit files in the audit directory.

Type: SIGHUP

Value range: an integer ranging from 1 to 1048576

Default value: **1048576**

> **NOTICE**
>
> Ensure that the value of this parameter is **1048576**. If the value is changed, the **audit_resource_policy** parameter may not take effect. To control the storage space and time of audit logs, use the **audit_resource_policy**, **audit_space_limit**, and **audit_file_remain_time** parameters.

16.22.2 Operation Audit

audit_operation_exec

Parameter description: Specifies whether to audit successful operations in GaussDB(DWS). Set this parameter as required.

Type: SIGHUP

Value range: a string

- **none**: indicates that no audit item is configured. If any audit item is configured, **none** becomes invalid.

- **all**: indicates that all successful operations are audited. This value overwrites the concurrent configuration of any other audit items. Note that even if this parameter is set to **all**, not all DDL operations are audited. You need to control the object level of DDL operations by referring to **audit_system_object**.

- **login**: indicates that successful logins are audited.

- **logout**: indicates that user logouts are audited.

- **database_process**: indicates that database startup, stop, switchover, and recovery operations are audited.

- **user_lock**: indicates that successful locking and unlocking operations are audited.

- **grant_revoke**: indicates that successful granting and reclaiming of a user's permission are audited.

- **ddl**: indicates that successful DDL operations are audited. DDL operations are controlled at a fine granularity based on operation objects. Therefore, **audit_system_object** is used to control the objects whose DDL operations are to be audited. (The audit function takes effect as long as **audit_system_object** is configured, no matter whether **ddl** is set.)

- **select**: indicates that successful SELECT operations are audited.

- **copy**: indicates that successful COPY operations are audited.

- **userfunc**: indicates that successful operations for user-defined functions, stored procedures, and anonymous blocks are audited.

- **set**: indicates that successful SET operations are audited.

- **transaction**: indicates that successful transaction operations are audited.

- **vacuum**: indicates that successful VACUUM operations are audited.

- **analyze**: indicates that successful ANALYZE operations are audited.

- **explain**: indicates that successful EXPLAIN operations are audited.

- **specialfunc**: indicates that successful calls to special functions are audited. Special functions include **pg_terminate_backend** and **pg_cancel_backend**.
- **insert**: indicates that successful INSERT operations are audited.
- **update**: indicates that successful UPDATE operations are audited.
- **delete**: indicates that successful DELETE operations are audited.
- **merge**: indicates that successful MERGE operations are audited.
- **show**: indicates that successful SHOW operations are audited.
- **checkpoint**: indicates that successful CHECKPOINT operations are audited.
- **barrier**: indicates that successful BARRIER operations are audited.
- **cluster**: indicates that successful CLUSTER operations are audited.
- **comment**: indicates that successful COMMENT operations are audited.
- **cleanconn**: indicates that successful CLEANCONNECTION operations are audited.
- **prepare**: indicates that successful PREPARE, EXECUTE, and DEALLOCATE operations are audited.
- **constraints**: indicates that successful CONSTRAINTS operations are audited.
- **cursor**: indicates that successful cursor operations are audited.

Default value: **login**, **logout**, **database_process**, **user_lock**, **grant_revoke**, **set**, **transaction**, and **cursor**

NOTICE

- You are advised to reserve **transaction**. Otherwise, statements in a transaction will not be audited.
- You are advised to reserve **cursor**. Otherwise, the **SELECT** statements in a cursor will not be audited.
- The Data Studio client automatically encapsulates **SELECT** statements using **CURSOR**.

---

audit_operation_error

Parameter description: Specifies whether to audit failed operations in GaussDB(DWS). Set this parameter as required.

Type: SIGHUP

Value range: a string

- **none**: indicates that no audit item is configured. If any audit item is configured, **none** becomes invalid.
- **syn_success**: synchronizes the [audit_operation_exec](#) configuration. To be specific, if the audit of a successful operation is configured, the corresponding failed operation is also audited. Note that even after **syn_success** is configured, you can continue to configure the audit of other failed operations. If **audit_operation_exec** is set to **all**, all failed operations are audited. If **audit_operation_exec** is set to **none**, **syn_success** is equivalent to **none**, that is, no audit item is configured.

- **parse**: indicates that the failed command parsing is audited, including the timeout of waiting for a command execution.

- **login**: indicates that failed logins are audited.

- **user_lock**: indicates that failed locking and unlocking operations are audited.

- **violation**: indicates that a user's access violation operations are audited.

- **grant_revoke**: indicates that failed granting and reclaiming of a user's permission are audited.

- **ddl**: indicates that failed DDL operations are audited. DDL operations are controlled at a fine granularity based on operation objects and configuration of **audit_system_object**. Therefore, failed DDL operations of the type specified in **audit_system_object** will be audited after **ddl** is configured.

- **select**: indicates that failed SELECT operations are audited.

- **copy**: indicates that failed COPY operations are audited.

- **userfunc**: indicates that failed operations for user-defined functions, stored procedures, and anonymous blocks are audited.

- **set**: indicates that failed SET operations are audited.

- **transaction**: indicates that failed transaction operations are audited.

- **vacuum**: indicates that failed VACUUM operations are audited.

- **analyze**: indicates that failed ANALYZE operations are audited.

- **explain**: indicates that failed EXPLAIN operations are audited.

- **specialfunc**: indicates that failed calls to special functions are audited. Special functions include **pg_terminate_backend** and **pg_cancel_backend**.

- **insert**: indicates that failed INSERT operations are audited.

- **update**: indicates that failed UPDATE operations are audited.

- **delete**: indicates that failed DELETE operations are audited.

- **merge**: indicates that failed MERGE operations are audited.

- **show**: indicates that failed SHOW operations are audited.

- **checkpoint**: indicates that failed CHECKPOINT operations are audited.

- **barrier**: indicates that failed BARRIER operations are audited.

- **cluster**: indicates that failed CLUSTER operations are audited.

- **comment**: indicates that failed COMMENT operations are audited.

- **cleanconn**: indicates that failed CLEANCONNECTION operations are audited.

- **prepare**: indicates that failed PREPARE, EXECUTE, and DEALLOCATE operations are audited.

- **constraints**: indicates that failed CONSTRAINTS operations are audited.

- **cursor**: indicates that failed cursor operations are audited.

- **blacklist**: indicates that the blacklist execution failure is audited.

Default value: **login**

audit_inner_tool

Parameter description: Specifies whether to audit the operations of the internal maintenance tool in GaussDB(DWS).

Type: SIGHUP

Value range: Boolean

- **on**: indicates that all operations of the internal maintenance tool are audited.
- **off**: indicates that all operations of the internal maintenance tool are not audited.

Default value: **off**

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 16-4**.

Default value: **12303**

Table 16-4 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.

Binary Bit	Meaning	Value Description
Bit 2	Whether to audit the CREATE, DROP, and ALTER operations on users.	• **0** indicates that the CREATE, DROP, and ALTER operations on these objects are not audited. • **1** indicates that the CREATE, DROP, and ALTER operations on these objects are audited.
Bit 3	Whether to audit the CREATE, DROP, ALTER, and TRUNCATE operations on tables.	• **0** indicates that the CREATE, DROP, ALTER, and TRUNCATE operations on these objects are not audited. • **1** indicates that the CREATE, DROP, ALTER, and TRUNCATE operations on these objects are audited.
Bit 4	Whether to audit the CREATE, DROP, and ALTER operations on indexes.	• **0** indicates that the CREATE, DROP, and ALTER operations on these objects are not audited. • **1** indicates that the CREATE, DROP, and ALTER operations on these objects are audited.
Bit 5	Whether to audit the CREATE, DROP, and ALTER operations on views.	• **0** indicates that the CREATE and DROP operations on these objects are not audited. • **1** indicates that the CREATE and DROP operations on these objects are audited.
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.

Binary Bit	Meaning	Value Description
Bit 9	Whether to audit the CREATE, DROP, and ALTER operations on resource pools.	• **0** indicates that the CREATE, DROP, and ALTER operations on these objects are not audited. • **1** indicates that the CREATE, DROP, and ALTER operations on these objects are audited.
Bit 10	Whether to audit the CREATE, DROP, and ALTER operations on workloads.	• **0** indicates that the CREATE, DROP, and ALTER operations on these objects are not audited. • **1** indicates that the CREATE, DROP, and ALTER operations on these objects are audited.
Bit 11	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 and DROP operations on these objects are not audited. • **1** indicates that the CREATE and DROP 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.

Binary Bit	Meaning	Value Description
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.
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**

enable_copy_server_files

Parameter description: Specifies whether to enable the permission to copy server files.

Type: POSTMASTER

Value range: Boolean

- **on** indicates that the permission to copy server files is enabled.
- **off** indicates that the permission to copy server files is disabled.

Default value: **true**

> **NOTICE**
>
> **COPY FROM**/**TO** *file* requires system administrator permissions. However, if the separation of permissions is enabled, system administrator permissions are different from initial user permissions. In this case, you can use **enable_copy_server_file** to control the **COPY** permission of system administrators to prevent escalation of their permissions.

16.23 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.

16.24 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 16-2**.

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**

16.25 Miscellaneous Parameters

enable_cluster_resize

Parameter description: If an SQL statement involves tables belonging to different groups, you can enable this parameter to push the execution plan of the statement to improve performance.

Type: SUSET

Value range: Boolean

- **on** indicates the execution plan of the statement can be pushed.
- **off** indicates the execution plan of the statement cannot be pushed.

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**

remote_read_mode

Parameter description: When **enable_crc_check** is set to **on** and the data read by the primary DN fails the verification, **remote_read_mode** is used to specify whether to enable remote read and whether to use secure authentication for connection upon the data verification failure. The setting takes effect only after the cluster is restarted.

Type: POSTMASTER

Value range: **off**, **non_authentication**, **authentication**

- **off**: indicates that the remote read function is disabled.
- **non_authentication**: indicates that the standby DN is connected and data is obtained when non-authentication is used.

- **authentication**: indicates that the standby DN is connected and data is obtained through authentication. Before restarting the cluster, ensure that a certificate exists in the **$GAUSSHOME/share/sslcert/grpc/** directory. Otherwise, the cluster cannot be started.

  Default value: **non_authentication**

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 16-5** 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 16-5 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**.	ORA TD

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

Configuration Item	Behavior	Applicable Compatibility Mode
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
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

Configuration Item	Behavior	Applicable Compatibility Mode
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
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

Configuration Item	Behavior	Applicable Compatibility Mode

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

Configuratio n Item	Behavior	Applicabl e Compati bility Mode

strict_concat_f unctions	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).
	
	<pre>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)</pre>
	When this parameter is specified, **NULL** is returned if any of the **textcat()**, **textanycat()**, and **anytextcat()** functions returns a null value.
	
	<pre>td_compatibility_db=# SELECT textcat('abc', NULL);
	textcat
	--------- 	
	
	(1 row)
	td_compatibility_db=# SELECT 'abc' || NULL;
	?column?
	---------- 	
	
	(1 row)</pre>
		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.
	<pre>mysql_compatibility_db=# SELECT LOCATE('sub', 'Substr');
	 locate
	-------- 	
	      0
	(1 row)</pre>
	• If this parameter is configured, the input parameter is case-insensitive.
	<pre>mysql_compatibility_db=# SELECT LOCATE('sub', 'Substr');
	 locate
	-------- 	
	      1
	(1 row)</pre>
		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.
	<pre>mysql_compatibility_db=# SELECT greatest(1, 2, null), least(1, 2, null);
	 greatest
		least
		----------+-------
		          |
		(1 row)</pre>● If this parameter is configured, the maximum or minimum value of non-null parameters is returned.
		<pre>mysql_compatibility_db=# SELECT greatest(1, 2, null), least(1, 2, null);
		 greatest
			least
			----------+-------
			        2
				1
				(1 row)</pre>
					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.
	<pre>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)</pre>
	● If this parameter is configured and pos is ≤ 0, characters are truncated from the left.
	<pre>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)</pre>
		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

Configuratio n Item	Behavior	Applicabl e Compati bility 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_i nline	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_afte r_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 = 'mysql_enable_division_by_zero';`
	`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.\nCREATE TABLE t1(c1 INT, c2 INT, c3 INT) DISTRIBUTE BY HASH(c1);\n-- Create a foreign table with the same structure in the local cluster.\nCREATE SERVER server_remote FOREIGN DATA WRAPPER gc_fdw options(ADDRESS 'address', DBNAME 'dbname', USERNAME 'username', PASSWORD 'password');\nCREATE FOREIGN TABLE t1(c1 INT, c2 INT, c3 INT) SERVER server_remote;\n-- Enable the parameter and see the pushdown behavior.\nSET behavior_compat_options = 'disable_gc_fdw_filter_partial_pushdown';\nEXPLAIN (verbose on,costs off) SELECT * FROM t1 WHERE c1>3 AND c2 <100 AND now() - '20230101' < c3;\n QUERY PLAN\n---\n---\n------\n Streaming (type: GATHER)\n Output: c1, c2, c3\n Node/s: All datanodes\n -> Foreign Scan on ca_schema.t1\n Output: c1, c2, c3\n Filter: ((t1.c1 > 3) AND (t1.c2 < 100) AND ((now() - '2023-01-01 00:00:00-08'::timestamp with time zone) < (t1.c3)::interval))\n Remote SQL: SELECT c1, c2, c3 FROM ca_schema.t1\n(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.\nSET behavior_compat_options = '';\nEXPLAIN (verbose on,costs off) SELECT * FROM t1 WHERE c1>3 AND c2 <100 AND now() - '20230101' < c3;\n 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)	

redact_compat_options

Parameter description: Specifies the compatibility option for calculation using masked data. This parameter is supported by version 8.1.3.310 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**

max_cache_partition_num

Parameter description: Specifies the number of memory-saving partitions in column-store mode during redistribution after scale-out. If the number of partitions exceeds the upper limit, the earliest cached partition is directly written to the column-store file.

Type: SIGHUP

Value range: an integer ranging from **0** to **32767**.

- **0** indicates that the memory-saving mode is disabled in column storage.
- Values from **1** to **32767** indicate the maximum number of partitions that can be cached in a partitioned table.

Default value: **0**

NOTE

This parameter is used for redistribution during scale-out. A proper value can reduce the memory consumption during redistribution of a partitioned column-store table. However, tables with unbalanced data distribution in some partitions may generate a large number of small CUs after the redistribution. If there are a large number of small CUs, execute the **VACUUM FULL** statement to merge them.

enable_prevent_job_task_startup

Parameter description: Specifies whether to prevent the thread startup of scheduled jobs. This is an internal parameter. You are not advised to change the value of this parameter.

Type: SIGHUP

Value range: Boolean

- **on**: Threads of scheduled jobs will not be started.
- **off**: Threads of scheduled jobs will be started.

Default value: **off**

NOTE

Set this parameter only on CNs.

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.

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.

object_mtime_record_mode

Parameter description: Sets the update action of the **mtime** column in the **PG_OBJECT** system catalog.

Type: SIGHUP

Value range: a string

- **default**: **ALTER**, **COMMENT**, **GRANT/REVOKE**, and **TRUNCATE** operations update the **mtime** column by default.

- **none**: The **mtime** column is not updated.

- **disable_acl**: **GRANT** or **REVOKE** operation does not update the **mtime** column.

- **disable_truncate**: **TRUNCATE** operations do not update the **mtime** column.

- **disable_partition**: Partition **ALTER** operations do not update the **mtime** column.

Default value: **default**

17 Glossary

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.
MPP	Massive Parallel Processing (MPP) refers to cluster architecture that consists of multiple machines. The architecture is also called a cluster system.

Term	Description
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.
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.
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.

Term	Description
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.
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.

Term	Description
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.
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.

Term	Description
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.
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.

Term	Description
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.
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.

Term	Description
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.
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).

Term	Description
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.
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.

Term	Description
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.