Data Warehouse Service

Troubleshooting

Issue 26

Date 2024-03-07

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Trademarks and Permissions

Notice

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

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

Contents

1 Database Connections

1.1 What Do I Do If gsql: command not found Is Displayed When I Run gsql to Connect to the Database?

Symptom

The following error information is displayed when running the **gsql -d postgres -p 26000 -r** command:

```
gsql: command not found...
```

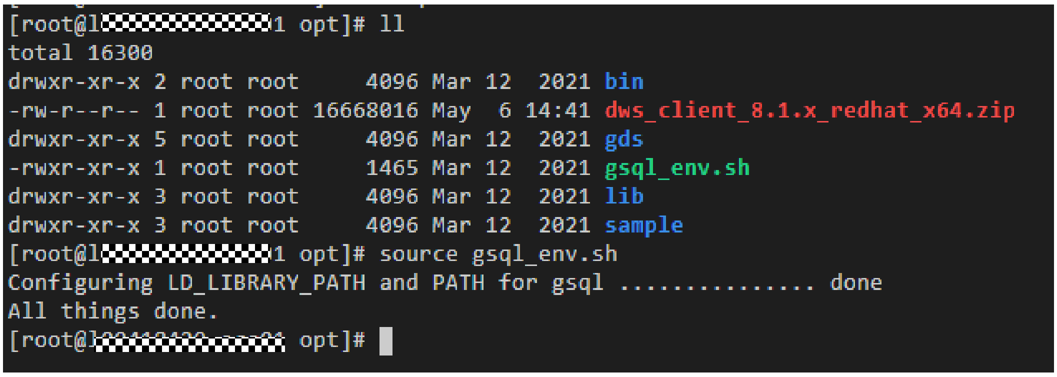
Possible Causes

- The command is not executed in the **bin** directory of gsql.
- The environment variable is not executed.

Handling Procedure

Step 1 Run the environment variable in the client directory, for example, in the **/opt** directory.

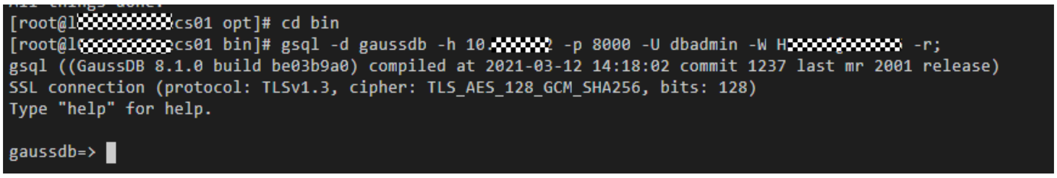
```
cd /opt
source gsql_env.sh
```



```
[root@[redacted] opt]# ll
total 16300
drwxr-xr-x 2 root root     4096 Mar 12  2021 bin
-rw-r--r-- 1 root root 16668016 May  6 14:41 dws_client_8.1.x_redhat_x64.zip
drwxr-xr-x 5 root root     4096 Mar 12  2021 gds
-rwxr-xr-x 1 root root     1465 Mar 12  2021 gsql_env.sh
drwxr-xr-x 3 root root     4096 Mar 12  2021 lib
drwxr-xr-x 3 root root     4096 Mar 12  2021 sample
[root@[redacted] opt]# source gsql_env.sh
Configuring LD_LIBRARY_PATH and PATH for gsql ............... done
All things done.
[root@[redacted] opt]#
```

Step 2 Go to the **bin** directory of gsql and run the **gsql** command to connect to the database.

```
cd bin
gsql -d gaussdb -h Database_IP_address -p 8000 -U dbadmin -W Database_user_password -r;
```



```
[root@l[redacted]cs01 opt]# cd bin
[root@l[redacted]cs01 bin]# gsql -d gaussdb -h 10.[redacted] -p 8000 -U dbadmin -W H[redacted] -r;
gsql ((GaussDB 8.1.0 build be03b9a0) compiled at 2021-03-12 14:18:02 commit 1237 last mr 2001 release)
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_128_GCM_SHA256, bits: 128)
Type "help" for help.

gaussdb=>
```

----End

1.2 Database Cannot Be Connected Using the gsql Client

Symptom

Users fail to connect to the database by running gsql on the client.

Possible Causes

- The number of system connections exceeds the upper limit. The following error information is displayed:

  ```
  gsql -d human_resource -h 10.168.0.74 -U user1 -p 8000 -W password -r
  gsql: FATAL:  sorry, too many clients already
  ```

- Users do not have the access permission to the database. The following error information is displayed:

  ```
  gsql -d human_resource -h 10.168.0.74 -U user1 -p 8000 -W password -r
  gsql: FATAL:  permission denied for database "human_resource"
  DETAIL:  User does not have CONNECT privilege.
  ```

- The network connection fails.

Solution

- The number of system connections exceeds the upper limit.

  You can set **max_connections** on the GaussDB(DWS) console.

  Set **max_connections**:

  a. Log in to the GaussDB(DWS) management console.

  b. In the navigation tree on the left, click **Clusters**.

  c. In the cluster list, find the target cluster and click its name. The **Basic Information** page is displayed.

  d. Click the **Parameter Modifications** tab and modify the value of parameter **max_connections**. Then click **Save**.

  e. In the **Modification Preview** dialog box, confirm the modification and click **Save**.

  You can check the number of connections as described in **Table 1-1**.

Table 1-1 Viewing the numbers of connections

Description	Command
View the upper limit of a user's connections.	Run the following command to view the upper limit of user **user1**'s connections. **-1** indicates that no connection upper limit is set for user **user1**. SELECT ROLNAME,ROLCONNLIMIT FROM PG_ROLES WHERE ROLNAME='user1'; rolname | rolconnlimit ---------+-------------- user1 | -1 (1 row)
View the number of connections that have been used by a specified user.	Run the following command to view the number of connections that have been used by user **user1**. **1** indicates the number of connections that have been used by user **user1**. SELECT COUNT(*) FROM V$SESSION WHERE USERNAME='user1'; count ------- 1 (1 row)
View the upper limit of connections to database.	Run the following command to view the upper limit of connections used by . **-1** indicates that no upper limit is set for the number of connections that have been used by . SELECT DATNAME,DATCONNLIMIT FROM PG_DATABASE WHERE DATNAME=''; datname | datconnlimit ----------+-------------- | -1 (1 row)
View the number of connections that have been used by a database.	Run the following commands to view the number of session connections that have been used by . **1** indicates the number of session connections that have been used by . SELECT COUNT(*) FROM PG_STAT_ACTIVITY WHERE DATNAME=''; count ------- 1 (1 row)
View the total number of connections that have been used by all users.	Run the following command to view the number of connections that have been used by all users: SELECT COUNT(*) FROM PG_STAT_ACTIVITY; count ------- 10 (1 row)

- Users do not have the access permission to the database.
  a. Connect to the database as user **dbadmin**.

  ```
  gsql -d human_resource -h 10.168.0.74 -U dbadmin -p 8000 -W password -r
  ```

  b. Grant the users with the access permission to the database.

  ```
  GRANT CONNECT ON DATABASE human_resource TO user1;
  ```

NOTE

Common misoperations may also cause a database connection failure, for example, entering an incorrect database name, user name, or password. In this case, the client tool will display the corresponding error messages.

```
gsql -d human_resource -p 8000
gsql: FATAL:  database "human_resource" does not exist

gsql -d human_resource -U user1 -W password -p 8000
gsql: FATAL:  Invalid username/password,login denied.
```

- The network connection fails.

  Check the network connection between the client and the database server. If you cannot ping from the client to the database server, the network connection is abnormal. Contact technical support.

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

1.3 An Error Indicating Too Many Client Connections Is Reported When a User Connects to a GaussDB(DWS) Database

Symptom

An error indicating too many client connections is reported when a user connects to a GaussDB(DWS) database.

- When a user uses an SQL client tool, such as gsql, to connect to a database, the following error information is displayed:

```
FATAL:  Already too many clients, active/non-active/reserved: 5/508/3.
```

- When the user uses multiple clients to concurrently connect to the database, the following error information is displayed:

```
[2019/12/25 08:30:35] [ERROR] ERROR: pooler: failed to create connections in parallel mode for thread 140530192938752, Error Message: FATAL:  dn_6001_6002: Too many clients already, active/non-active: 468/63.
FATAL:  dn_6001_6002: Too many clients already, active/non-active: 468/63.
```

Possible Causes

1. **The number of current database connections exceeds the upper limit.**

   In the error information, the value of **non-active** indicates the number of idle connections. For example, if the value of **non-active** is 508, there are 508 idle connections.

2. **The upper limit of connections is set when the user is created.**

   If the number of connections does not reach the upper limit, the possible cause is that the maximum number of connections is set when the user is created.

Handling Procedure

You can preferentially use the following methods to rectify the fault:

1. Release all **non-active** connections temporarily.
   ```
   SELECT PG_TERMINATE_BACKEND(pid) from pg_stat_activity WHERE state='idle';
   ```
2. On the GaussDB(DWS) management console, set parameter **session_timeout**, which controls the timeout period of idle sessions. After an idle session's timeout period exceeds the specified value, the server automatically closes the connection.

   The default value of parameter **session_timeout** is **600** seconds. The value **0** indicates that the timeout limit is disabled. You are not advised to set **session_timeout** to **0**.

   The procedure for setting parameter **session_timeout** is as follows:

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

Step 2 In the navigation tree on the left, click **Cluster Management**.

Step 3 In the cluster list, find the target cluster and click its name. The **Basic Information** page is displayed.

Step 4 Click the **Parameter Modifications** tab and modify the value of parameter **session_timeout**. Then click **Save**.

Step 5 In the **Modification Preview** dialog box, confirm the modification and click **Save**.

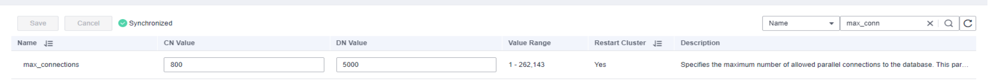
----End

If the preceding methods cannot meet service requirements, perform the following operations:

If the number of database connections exceeds the maximum, perform the following operations:

1. Check where the connections on the CN come from, the total number of connections, and whether the number of connections exceeds the value of **max_connections**. The default value is **800** for CNs and **5000** for DNs.
   ```
   SELECT coorname, client_addr, count(1) FROM pgxc_stat_activity group by coorname, client_addr
   order by coorname;
   ```
2. Check whether the value of **max_connections** can be increased. The adjustment policies are as follows:
   - Increasing the value of **max_connections** for CNs will increase the number of concurrent queries connected to DNs. Therefore, you need to increase the values of **max_connections** and **comm_max_stream** for DNs,
   - Increase the max_connections value of CNs/DNs by two times. For clusters with small values, increase it by four times.
   - To avoid failures in the preparation step, the value of **max_prepared_transactions** cannot be smaller than that of **max_connections**. You are advised to set **max_prepared_transactions** to a value equal to that of **max_connections**. In this way, each session can have a prepared transaction in waiting state.
3. Change the value of **max_connections** on the management console.

On the management console, choose **Basic Information**, click the **Parameter Modification** tab, change the value of **max_connections**, and click **Save**.



Name	CN Value	DN Value	Value Range	Restart Cluster	Description
max_connections	800	5000	1 - 262,143	Yes	Specifies the maximum number of allowed parallel connections to the database. This par...

If the maximum number of connections of a user is set, perform the following operations:

The value is specified by the CONNECTION LIMIT connlimit parameter of the CREATE ROLE command used when a user is created. After the value is specified, you can also change it through the CONNECTION LIMIT connlimit parameter of the ALTER ROLE command.

1. Use **PG_ROLES** to check the maximum number of connections of a specified user.

```
SELECT ROLNAME,ROLCONNLIMIT FROM PG_ROLES WHERE ROLNAME='role1';
 rolname | rolconnlimit
---------+--------------
 role1   |           10
(1 row)
```

2. Change the maximum number of connections of a user.

```
ALTER ROLE role1 connection limit 20;
```

1.4 Cluster IP Address Cannot Be Pinged/Accessed

Symptom

The client host cannot ping the access address of a GaussDB(DWS) cluster.

Possible Causes

- **The network is disconnected.**

  If the client host connects to a GaussDB(DWS) cluster using the cluster's private IP address, check whether the client host and the GaussDB(DWS) cluster are in the same VPC and subnet. If they are not in the same VPC or subnet, the network is disconnected.

- **Ping is not allowed in the security group rule.**

  The cluster's address can be pinged only when the ICMP port is enabled in the inbound rule of the cluster's security group. By default, only **TCP and port 8000** are enabled in the security group that is automatically created during GaussDB(DWS) cluster creation.

  If the ICMP port is enabled in the inbound rule of the security group, check whether the source address in the inbound rule covers the IP address of the client host. If not, the client host cannot ping the cluster.

Handling Procedure

- **The network is disconnected.**

  If the client host connects to the data warehouse cluster using the cluster's private IP address, apply for another ECS as the client host. The ECS must be in the same VPC and subnet as the data warehouse cluster.

- **Ping is not allowed in the security group rule.**

  View the cluster's security group rule to check whether it enables the ICMP port for the client host's IP address. The operations are as follows:

  a. Log in to the GaussDB(DWS) management console.

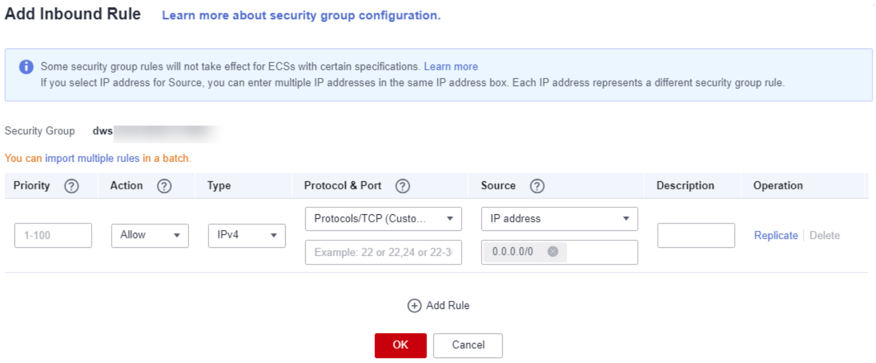
  b. On the **Cluster Management** page, find the target cluster and click its name. The **Basic Information** page is displayed.

  c. Locate the **Security Group** parameter and click the security group name to switch to the **Security Groups** page on the VPC console.

  d. Click the **Inbound Rules** tab and check whether an inbound rule that enables the ICMP port exists. If such an inbound rule does not exist, click **Add Rule** to add one.

     - **Protocol & Port**: Select **ICMP** and **All**.
     - **Source**: Select **IP address**, and enter the IP address and subnet mask of the client host. **0.0.0.0/0** indicates any IP address.

Figure 1-1 Inbound rules



Add Inbound Rule Learn more about security group configuration.

Some security group rules will not take effect for ECSs with certain specifications. Learn more
If you select IP address for Source, you can enter multiple IP addresses in the same IP address box. Each IP address represents a different security group rule.

Security Group dws[redacted]

You can import multiple rules in a batch.

Priority	Action	Type	Protocol & Port	Source	Description	Operation
1-100	Allow	IPv4	Protocols/TCP (Custo... Example: 22 or 22,24 or 22-3	IP address 0.0.0.0/0		Replicate | Delete

  e. Click **OK**.

1.5 Error "An I/O error occurred while sending to the backend" Is Reported During Service Execution

Symptom

Error "An I/O error occurred while sending to the backend" is reported during service execution on a client connected to GaussDB(DWS).

Possible Causes

The client is disconnected from the database. The disconnection may be one of the following two cases:

- Disconnected at the database side.

  After the database is disconnected, if there are applications depend on the connection running, this error will be reported. The possible causes are as follows:

   a. The CN process restarts unexpectedly.
   b. The session times out.
   c. A user manually runs a command to terminate the session.

- Disconnected at the client side.

Handling Procedure

If the disconnection occurs at the database side. The handling methods corresponding to the preceding causes are as follows:

1. Run the following command to check whether the CN process restarts unexpectedly:

   ```
   ps -eo pid,lstart,etime,cmd | grep coo
   ```

   If the connection exists before the CN process is started, the connection will be disconnected after the CN process is restarted. If the service side continues to use the connection, an error is reported.
2. The parameter **session_timeout** is set. The default value is **10 minutes**. The value **0** indicates that timeout is disabled. When the session time exceeds the value of this parameter, the database automatically clears the connection. In scenarios that require long time connections. You can set **session_timeout** to the expected duration on the client. For details, see **JDBC Error Occurs Due to session_timeout Settings**.
3. Check whether CN logs contain "due to".

   Check whether a user manually runs the **select pg_terminate_backend(pid);** command to terminate the session based on the corresponding time in the log. In most cases, the session is terminated due to user operations.

   ```
   ERROR: dn_6003_6004: abort transaction due to concurrent update test 289502.
   FATAL: terminating connection due to administrator command
   ```

If the disconnection occurs at the client side:

If the disconnection is not at the database side, it may be at the client side.

1. Check whether the timeout parameter **socketTimeout** is set on the client. If yes, set this parameter to a proper value.
2. Check whether the disconnection is caused by network problems.

2 JDBC/ODBC

2.1 Locating JDBC Issues

Java Database Connectivity (JDBC) is a unified standard interface for applications to access databases. Applications can use JDBC to connect to databases and execute SQL statements. GaussDB(DWS) supports JDBC 4.0. This section describes how to locate common JCDB faults and rectify the faults.

The causes of JDBC problems are as follows:

1. Applications or application framework errors
2. JDBC function errors
3. Improper database configurations

JDBC problems can be classified into the following types:

1. An error is reported during execution, and JDBC throws an exception.
2. The execution duration is abnormally long.
3. A feature is not supported. The JDBC has not implemented the required JDK interface.

The following table describes JDBC issues in detail.

Table 2-1 JDBC issues

Type	Cause
Database Connection Fails	The JDBC client configuration is incorrect. For example, the URL format is incorrect, or the user name or password is incorrect.
	The network is abnormal.
	A JAR package conflict occurs.
	The database configuration is incorrect. The remote access permission has not been configured for the database.

Type	Cause
Service Execution Exceptions	The input SQL statement is incorrect and is not supported by GaussDB(DWS).
	An error occurs during service processing, and an abnormal packet is returned.
	Network fault.
	The database connection times out, and the socket is closed.
Performance Issues	Slow SQL execution
	The result set is too large. As a result, the application segment responds slowly.
	The passed SQL statement is too long and the JDBC parsing is slow.
Function Support Issues	The JDK does not provide standard APIs.
	The JDBC does not implement the APIs.

2.2 Database Connection Fails

Check that the hostname and port are correct and that the postmaster is accepting TCP/IP connections.

Possible cause: The network between the client and server is disconnected, the port is incorrect, or the CN to be connected is abnormal.

Solution:

- On the client, ping the IP address of the server to check whether the network connection is normal. If the network connection is abnormal, rectify the network fault.
- Check whether the port for connecting to the CN in the URL is correct. If the port is incorrect, correct it (the default port is 8000).

FATAL: Invalid username/password,login denied.

Possible cause: The username or password is incorrect.

Solution: Use the correct database username and password.

No suitable driver found for XXXX

Possible cause: The format of the URL used for establishing the JDBC connection is wrong.

Solution: Correct the URL format.

- The URL format of gsjdbc4.jar is **jdbc:postgresql://host:port/database**.
  - The version is 8.1.x when the pom dependency is used.
- The URL format of gsjdbc200.jar is **jdbc:gaussdb://host:port/database**.
  - The version is 8.1.x-200 when the pom dependency is used.

conflict

Possible cause: The JDBC JAR package conflicts with an application. For example, JDBC and an application have classes with the same path and name.

- **gsjdbc4.jar** conflicts with the open-source **postgresql.jar** because they have the same class name.
- **gsjdbc4.jar** included other tools, such as **fastjson**, required for IAM features. This tool conflicts with **fastjson** in an application.

Solution:

- For the open-source **postgresql.jar** file, use **gsjdbc200.jar** instead and use different URL format and driver path. Change the driver name from **org.postgresql.Driver** to **com.huawei.gauss200.jdbc.Driver**, and change the URL format from **org:postgresql://host:port/database** to **jdbc:gaussdb://host:port/database**.
- If the JAR package introduced by JDBC conflicts with that introduced by an application, use the shade of Maven to modify the class path in the JAR package.
- Check whether the JDBC driver is **gsjdbc4.jar** or **gsjdbc200.jar**. If **gsjdbc4.jar** is used, replace it with **gsjdbc200.jar** and try to establish a connection.

  NOTE

  For the pom dependency, replace the 8.1.x version with the 8.1.x-200 version.

org.postgresql.util.PSQLException: FATAL: terminating connection due to administrator command Session unused timeout

Possible cause: The connection is disconnected due to session timeout.

Solution: Check the timeout configuration on the CN and the JDBC client. Increase the timeout interval or disable the timeout configuration.

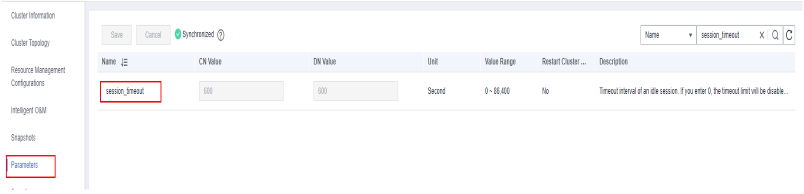
1. Check the CN logs where the error is reported. If the log contains **session unused timeout**, there is session timeout.

   Solution:

   a. Log in to the DWS console and click the name of the target cluster.

   b. On the displayed page, click the **Parameters** tab and search for **session_timeout** to view the timeout interval.

   c. Set the values of **session_timeout** on CNs and DNs to **0**. For details, see "Modifying Database Parameters" in the *GaussDB(DWS) User Guide*.



Name	CN Value	DN Value	Unit	Value Range	Restart Cluster ...	Description
session_timeout	600	600	Second	0 ~ 86,400	No	Timeout interval of an idle session. If you enter 0, the timeout limit will be disable...

Connection refused: connect.

Possible cause: The default driver of the third-party tool is incompatible.

Solution: Replace the JDBC driver package and check whether the connection is normal.

Connections could not be acquired from the underlying database!

Possible causes:

- The driver setting is incorrect.
- The database connection address is correct.
- The password or account is incorrect.
- The database is not started or you do not have the permission to access the database.
- The required driver JAR package is not imported to the project.

Solution:

- Check the driver configuration and correct it.
  - gsjdbc4.jar driver=org.postgresql.Driver
  - gsjdbc200.jar driver=com.huawei.gauss200.jdbc.Driver
- Check the database connection address and correct it.
  - For **gsjdbc4.jar**, the format is **jdbc:postgresql://host:port/database**.
  - For **gsjdbc200.jar**, the format is **jdbc:gaussdb://host:port/database**.
- Use the correct database username and password.
- Check whether the database is started or the access permission has been obtained.
- Check whether the used JDBC driver is **gsjdbc4.jar** or **gsjdbc200.jar**. Use the correct JDBC driver JAR package.
  - **gsjdbc4.jar**: The **gsjdbc4.jar** driver package is compatible with PostgreSQL. Its class names and class structures are the same as those of the PostgreSQL driver. PostgreSQL applications can be directly migrated to the current system.
  - **gsjdbc200.jar**: If a JVM process needs to access PostgreSQL and GaussDB(DWS) at the same time, this driver package must be used. In this package, the main class name is **com.huawei.gauss200.jdbc.Driver** (replace **org.postgresql** with **com.huawei.gauss200.jdbc**). The URL prefix of the database connection is **jdbc:gaussdb**. Other parameters are the same as those of **gsjdbc4.jar**.

The JDBC DEV environment is normal. The test environment cannot be connected and a null pointer error or a URI error "uri is not hierarchical" is reported.

Problem analysis: Some virtual environments do not support the function of obtaining extended parameters. Therefore, you need to disable the function

Solution: Add connection configuration **connectionExtraInfo=false**. For details, see.

```
jdbc:postgresql://host:port/database?connectionExtraInfo=false
```

An error is reported when the open-source JDBC SSL mode is used to connect to DWS

Possible cause: SSL full verification is not enabled to check whether the URL is completely matched when the open-source JDBC is used.

Solution: Add **sslmode=require** for the open-source connection.

```
jdbc:postgresql://host:port/database?sslmode=require
```

Connection cannot be converted to BaseConnection when the connection pool is used to obtain connections in the CopyManager scenario

Possible cause: BaseConnection is a non-public class. The connection pool object needs to be unwrapped to obtain the original PGConnection.

Solution: Unwrap the object and return the original object to allow access to non-public methods.

```
// Unwrap
PGConnection unwrap = connection.unwrap(PGConnection.class);
// Convert
BaseConnection baseConnection = (BaseConnection)unwrap;
CopyManager copyManager = new CopyManager(baseConnection);
```

2.3 Service Execution Exceptions

Broken pipe, connection reset by peer

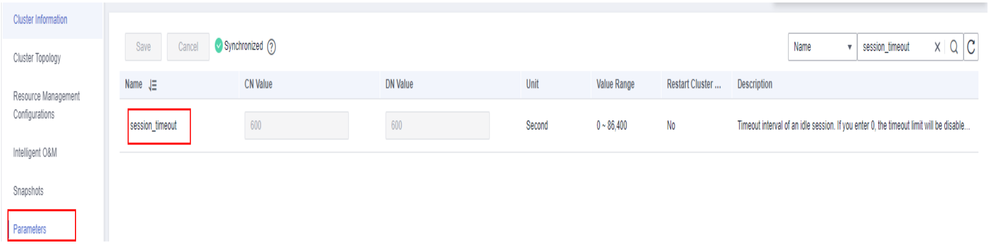
Possible cause: The network is faulty and the database connection times out.

Solution: Check the network status, rectify the network fault, and rectify the configuration related to database connection timeout, for example, the database parameter **session_timeout**.

Step 1 Log in to the DWS console and click the name of the target cluster.

Step 2 On the displayed page, click the **Parameters** tab and search for **session_timeout** to view the timeout interval.

Step 3 Set the values of **session_timeout** on CNs and DNs to **0**. For details, see "Modifying Database Parameters" in the *GaussDB(DWS) User Guide*.



Name	CN Value	DN Value	Unit	Value Range	Restart Cluster ...	Description
session_timeout	600	600	Second	0 ~ 86,400	No	Timeout interval of an idle session. If you enter 0, the timeout limit will be disable...

----End

The column index is out of range

Possible cause: The result set obtained by the application is not the expected one, and the number of columns is incorrect.

Solution: Check the database table definition and SQL statements. If the result set contains only three columns, the maximum value of **index** is **3**.

"Tried to send an out-of-range integer as a 2-byte value" Is Reported Due to Too Many Parameters in SQL Statements

Possible cause: According to the JDBC protocol, the total number of variables cannot exceed 32767, which is the maximum value of short Int.

Solution:

Data query: Split large SQL statements to ensure that the number of variables in each SQL statement is less than 32767.

Data import: Import data in batches or use copymanager .

Error "ERROR: cached plan must not change result type" Reported When the Stored Procedure Is Invoked

Possible cause: **PreparedStatement** is used in JDBC. By default, a plan is cached after being executed for five times. If there are table-creation operations after that (for example, the table definition modification), the error **ERROR: cached plan must not change result type** will be reported when the plan is executed again.

Solution: Set **prepareThreshold** to **0** in the JDBC connection string so that the plan is not cached. Example:

```
String url = "jdbc:postgresql:// 192.168.0.10:2000/postgres?prepareThreshold=0";
```

"ERROR: insufficient data left in message" Is Reported When JDBC Is Used to Execute SQL Statements

Possible cause: The server cannot process the '\0' character in the string. '\0' indicates the string whose value is 0x00 and '\u0000' in UTF encoding.

Solution: Check whether the SQL statement executed by the customer contains '\0'. If yes, replace it with a space.

"ERROR: relation xx already exists" Is Reported When the CREATE TABLE AS Statement Is Executed Using JDBC

Possible cause: When JDBC invokes **preparedStatement.getParameterMetaData()**, a P packet is sent. This packet creates a table in the database, which causes duplication during execution.

Solution: When using **preparedStatement**, you are advised to split the CREATE TABLE AS statement or use **resultSet.getMetaData()**.

2.4 Performance Issues

processResult is time-consuming

Set **loglevel** to **3** and enable the JDBC log function. There are two scenarios where the processResult phase is time-consuming:

1. JDBC waits a long time for the database to return packets.

   Possible cause: If the interval between the **FE=> Syncr** log and the **<=BE ParseComplete** log is long, the execution is slow.

   Solution: Analyze the cause of slow SQL execution. For details, see **SQL Execution Is Slow with Low Performance and Sometimes Does Not End After a Long Period of Time**.

2. It takes a long time when the result set is too large and all data is loaded at a time.

   Possible cause: View logs. If **<=BE DataRow** logs appear too many times or there are a large volume of query results returned by **select count(*)** command, the result set is too large.

   Solution: Set **fetchSize** to a small value so that data is returned in batches and the client can quickly respond.

   ```
   statement.setFetchSize(10);
   ```

modifyJdbcCall and createParameterizedQuery are time-consuming

Possible cause: If modifyJdbcCall (verifying specification of the passed SQL statements) and createParameterizedQuery (parsing the passed SQL statements into preparedQuery to obtain subqueries consisting of simplequery) take a long time, check whether the SQL statements need to be optimized.

Solution: Check whether the SQL statements can be optimized on the application side..

2.5 Function Support Issues

not yet implemented

Possible cause: APIs are not implemented in JDBC.

Solution: Technical personnel need to check whether the APIs can be implemented or whether other APIs provide the same function. Use the provided APIs.

JDK standard APIs do not support a feature.

Possible cause: The JDK does not provide an API.

Solution: If the JDK does not provide an API, the feature is not supported by JDBC. You can call public methods in JDBC classes to obtain data as needed.

3 Data Import and Export

3.1 "ERROR: invalid byte sequence for encoding 'UTF8': 0x00" Is Reported When Data Is Imported to GaussDB(DWS) Using COPY FROM

Symptom

"ERROR: invalid byte sequence for encoding 'UTF8': 0x00" is reported when data is imported to GaussDB(DWS) using COPY FROM.

Possible Causes

The data file is imported from an Oracle database, and the file is UTF-8 encoded. The error message also contains the number of lines. Because the file is too large to be opened by running the **vim** command, the **sed** command is used to extract the lines, and then the **vim** command is used to open the file. No exception is found. Part of the file can be imported after running the **split** command to split the file by the number of lines.

According to the analysis, fields or variables of the varchar type in GaussDB(DWS) cannot contain '\0' (that is, 0x00 and UTF encoding '\u0000'). Delete '\0' from the string before importing it.

Handling Procedure

Run the **sed** command to replace **0x00**.

```
sed -i 's/\x00//g;' file
```

Parameter:

- **-i** indicates replacement in the original file.
- **s/** indicates single replacement.
- **/g** indicates global replacement.

3.2 Data Import and Export Faults with GDS

When GDS is used to import or export data, character set problems may occur, especially when data is migrated across databases of different types or different encoding types. As a result, data import fails, severely blocking data migration and other onsite operations.

Locale Support

Locale support refers to cultural preference compliance of applications, including alphabetic, sorting, and number formats. A locale is automatically initialized when a database is created using **initdb**. By default, **initdb** initializes the database based on the locale of its execution environment. The locale is preset in the system. You can use **initdb –locale=***xx* to specify another locale.

If you need to use the rules of multiple locales, you can use the following locale categories to control the localization rules. These categories are converted to **initdb** options, indicating locale selections of a particular category.

Table 3-1 Locale support

Category	Description
LC_COLLATE	Defines character-collation or string-collation information.
LC_CTYPE	Defines character classification, for example, what counts as a character, case conversion, and others.
LC_MESSAGES	Defines the language used by messages.
LC_MONETARY	Defines formatting for monetary numeric information.
LC_NUMERIC	Defines formatting for non-monetary numeric information.
LC_TIME	Defines formatting for time and date information.

If you want the system to behave like there is no locale support, you can use the C locale or its equivalent the POSIX locale. Using non-C or non-POSIX locales may affect the performance, because it slows character processing and prevents the use of normal indexes in the LIKE clause. Therefore, use non-C or non-POSIX locales only when they are required.

Some locale category values must be fixed upon the creation of a database. Different databases can use different value settings. However, once a database is created, these locale category values cannot be changed in the database. For example, LC_COLLATE and LC_CTYPE require fixed values upon the creation of a database. These two locale categories determine the collation of indexes, so they must be fixed. Otherwise indexes in the text column will break down. Default values of these two locale categories are set when **initdb** is running, and will be used to create new databases, unless they are otherwise specified by the CREATE DATABASE command. Other locale category values can be changed at any time. To change a locale category value, set it to the configuration parameters of the server named the same as the locale category. The values chosen by **initdb** are written to the **postgresql.conf** file as the default values for the server startup. If you delete these values from the **postgresql.conf** file, the server will inherit the settings from its execution environment.

In particular, the locale setting affects the following SQL features:

- Data collation in the process of queries that use ORDER BY or a standard comparison operator on text data
- UPPER, LOWER, and INITCAP functions
- Pattern matching operators (such as LIKE, SIMILAR TO, and POSIX style regular expressions), case-insensitive matching, and character classification using character regular expressions
- TO_CHAR functions

Therefore, inconsistent query result sets in these scenarios are probably caused by the character set.

Collation Support

This feature allows you to specify the data collation and character classification behavior for each column or even each operation, which relaxes the restriction that the LC_COLLATE and LC_CTYPE settings cannot be changed since the database is created.

The collation of an expression can be a default rule that matches the locale of the database. The collation rule can also be uncertain. In this case, the collation operation and other operations that depend on the collation rule will fail.

When the database system must perform collation or character classification, it uses the collation rule of the input expression. This happens when you use the ORDER BY clause and functions, or when you call an operator (such as <). The collation rule used by the ORDER BY clause is the collation rule of the sort key. The collation rule used by a function or an operator call is derived from their parameters. In addition to comparison operators, functions (such as LOWER, UPPER, and INITCAP) that convert between uppercase and lowercase letters will refer to the collation rule. Pattern matching operators and TO_CHAR related functions also need to refer to the collation rule.

For a function or an operator call, it checks the collation parameter when the specified operation is performed, to obtain the collation rule. If the result of the function or operator call is a sortable data set, the collation rule is also used by the function or operator expression in case a peripheral expression requires the collation rule of the function or operator expression.

The collation derivation of an expression can be implicit or explicit. This distinction determines how collations are organized 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 organized, for example in a function call, the following rules are used:

1. If any input expression has an explicit collation derivation, all explicitly derived collations among the input expressions must be the same, otherwise an error is raised. If any explicitly derived collation exists, it is the result of the collation combination.
2. Otherwise, all input expressions must have the same implicit collation derivation or the default collation. If any non-default collation exists, that is the result of the collation combination. Otherwise, the result is the default collation.
3. If there are conflicting non-default implicit collations among the input expressions, the combination is deemed to have indeterminate collation. This is not an error unless the particular function called requires knowledge of the collation it should apply. If it does, an error will be raised at runtime.

Character Set Support

The character set support in PostgreSQL enables you to store text in various character sets (also called encodings), including single-byte character sets such as the ISO 8859 series and multiple-byte character sets such as EUC (Extended Unix Code), UTF-8, and Mule internal code. MPPDB mainly uses the GBK, UTF-8, and LATIN1 character sets. All supported character sets can be used transparently by clients, but a few are not supported for use within the server (that is, as a server-side encoding. GBK encoding in PostgreSQL database is only client-side encoding, not server-side encoding. MPPDB introduces GBK to server-side encoding, which is the root cause of many problems). The default character set is selected while initializing your PostgreSQL database using **initdb**. It can be overridden when you create a database, so you can have multiple databases each with a different character set. An important restriction is that each database's character set must be compatible with the database's LC_CTYPE (character classification) and LC_COLLATE (string sort order) locale settings. For the C or POSIX locale, any character set is allowed, but for other locales there is only one character set that can work correctly. On Windows, however, UTF-8 encoding can be used with any locale.

The SQL_ASCII setting behaves considerably differently from the other settings. When the server character set is SQL_ASCII, the server interprets byte values 0-127 according to the ASCII standard. Byte values 128-255 are deemed as uninterpreted characters. No encoding conversion will be done if the setting is SQL_ASCII. Therefore, this setting is not a declaration that a specific encoding is in use, as a declaration of ignorance about the encoding. In most cases, if you are working with non-ASCII data, it is unwise to use the SQL_ASCII setting because PostgreSQL is unable to help you convert or validate non-ASCII characters.

Which encoding is supported by a database system is determined by three aspects: database server support, database access interface support, and client support.

- Database server support

  Database server support means a server can receive character set in a certain encoding format and store the character set, and it can also provide the character set (including identifiers and character field values) to the client. In addition, it can convert the characters to other encoding formats, for example, from UTF-8 to GBK.

You can specify the database server encoding when creating a database: CREATE DATABASE ... ENCODING ... // ASCII, UTF-8, EUC_CN, and more are supported.

You can check database encoding: SHOW server_encoding

- Database access interface support

  Database access interface support means that the API must be able to correctly read and write the characters of a certain encoding format, without any data loss or distortion. The following uses the JDBC interface as an example:

  The JDBC interface sets client_encoding based on the file.encoding of JVM: set client_encoding to file_encoding

  Then converts a string to a byte stream encoded in the client_encoding format, and send the byte stream to the server. Prototype: **String.getBytes(client_encoding)**.

  After receiving the byte stream from the server, the client uses client_encoding to construct a string object as the return value of **getString** and send the object to the application. Prototype: **String(byte[], ..., client_encoding)**.

- Client support

  Client support means that the client can display the characters that are read from the database and can submit the characters to the server.

  You can specify the client encoding of a session: **SET CLIENT_ENCODING TO'value'**

  You can check database encoding: **Show client_encoding**

Solutions to Character Set Problems During GDS Import and Export

Problem 1: 0x00 characters cannot be saved to the database. **ERROR: invalid byte sequence for encoding "UTF8": 0x00**

Cause: The PostgreSQL database does not allow 0x00 characters in text data. This is a baseline problem. Other databases do not have this problem.

Solution:

1. Replace 0x00 characters.
2. The COPY command and GDS both have the **compatible_illegal_chars** option. If this option is enabled (the COPY command and GDS foreign table can be altered), single-byte or multi-byte invalid characters will be replaced with spaces or question marks (?). In this way, the data can be imported successfully, but the original data is changed.
3. Create a database whose encoding is SQL_ASCII and set **client_encoding** to SQL_ASCII (you can set it using the COPY command or in the GDS foreign table). In this case, special processing and conversion of the character set can be avoided, all sorting, comparison, and processing related to the library are processed as single bytes.

3.3 Failed to Create a GDS Foreign Table and An Error Is Reported Indicating that ROUNDROBIN Is Not Supported

Symptom

A GDS foreign table cannot be created and an error is reported indicating that **ROUNDROBIN** is not supported. The message is as follows:

```
ERROR:  For foreign table ROUNDROBIN distribution type is built-in support.
```

Possible Causes

By default, a GDS foreign table is created in **ROUNDROBIN** distribution mode. **ROUNDROBIN** distribution information cannot be explicitly added during table creation.

Handling Procedure

When creating a GDS foreign table, delete the specified distribution information, that is, delete the specified **DISTRIBUTE BY ROUNDROBIN** in the statement.

3.4 When CDM Is Used to Import MySQL Data to GaussDB(DWS), the Column Length Exceeds the Threshold and Data Synchronization Fails

Symptom

In MySQL 5.*x*, the column length is VARCHAR(*n*). When CDM is used to synchronize data to GaussDB(DWS) and the column length is set to VARCHAR(*n*), the column length exceeds the threshold and data synchronization fails.

Possible Causes

- In versions earlier than MySQL 5.0.3, *n* in VARCHAR(*n*) indicates the number of bytes.
- In MySQL 5.0.3 and later, *n* in VARCHAR(*n*) indicates the number of characters. For example, VARCHAR(200) indicates that a maximum of 200 English or Chinese characters can be stored.
- For GaussDB(DWS), *n* in VARCHAR(*n*) indicates the number of bytes.

Each GBK character occupies two bytes and each UTF-8 character occupies a maximum of three bytes. Based on the conversion rule, for the same column length, the length may exceed the threshold on GaussDB(DWS).

Handling Procedure

If the MySQL column is VARCHAR(*n*), set the column length of GaussDB(DWS) to VARCHAR(*n*3*).

3.5 "Access Denied" Is Displayed When the SQL Statement for Creating an OBS Foreign Table Is Executed

Symptom

When a user executes the SQL statement for creating an OBS foreign table, an OBS error **Access Denied** is reported.

Possible Causes

- If the AK and SK in the statement for creating an OBS foreign table are incorrect, the following error information is displayed:

  ```
  ERROR: Fail to connect OBS in node:cn_5001 with error code: AccessDenied
  ```

- If an account does not have the read and write permissions on corresponding OBS buckets, the following error information is displayed:

  ```
  dn_6001_6002: Datanode 'dn_6001_6002' fail to read OBS object bucket:'obs-bucket-name'
  key:'xxx/xxx/xxx.csv' with OBS error code:AccessDenied message: Access Denied
  ```

  By default, an account does not have the permission to access OBS data of other accounts. In addition, an IAM user (similar to a sub-user) does not have the permission to access OBS data of the account to which it belongs.

Handling Procedure

- **The AK and SK in the statement for creating an OBS foreign table are incorrect.**

  Obtain the correct AK and SK and write them into the SQL statement. To obtain the AK and SK, perform the following steps:

  a. Log in to the GaussDB(DWS) management console.

  b. Move the cursor to the username in the upper right corner and choose **My Credentials**.

  c. In the navigation pane, click **Access Keys**.

     On the **Access Keys** page, you can view the existing access key ID (AK).

  d. If you want to obtain both the AK and SK, click **Create Access Key** to create and download the access key file.

- **The account does not have the read and write permissions on OBS buckets.**

  You must grant the required OBS access permissions to specified users.

  – When importing data to GaussDB(DWS) using an OBS foreign table, the user who performs the operation must have the read permission on the OBS bucket and object where the source data files are located.

- When exporting data using an OBS foreign table, the user who performs the operation must have the read and write permissions on the OBS bucket and object where the data export path is located.

For details about configuring OBS permissions, see **Configuring a Bucket ACL** and **Configuring Object ACL** in the *Object Storage Service Console Operation Guide*.

3.6 Disk Usage Increases After Data Fails to Be Imported Using GDS

Symptom

A user uses GDS to import data but fails, triggering data import again. After data is imported, the user checks the disk space and finds that newly occupied disk space is much greater than the imported data volume.

Cause Analysis

After data fails to be imported, the occupied disk space is not released.

Solution

Step 1 Check the logs of GDS import jobs to see whether any execution failure occurs.

Step 2 Clear the related table or partition.

```
vacuum [full] table_name;
```

----End

3.7 Error Message "out of memory" Is Displayed When GDS Is Used to Import Data

Symptom

When GDS is used to import data, the error message "out of memory" is displayed during script execution.

Possible Causes

1. When the **COPY** command is executed or data is imported, the size of a single row exceeds 1 GB.
2. The source file format is incorrect. For example, the quotation marks are unpaired. As a result, the size of a single row identified by the system exceeds 1 GB.

Handling Procedure

1. Ensure that the quotation marks in the source file appear in pairs.

2. Check whether the parameter values and formats in the commands for creating a foreign table are proper.
3. Check whether the size of the row in the error report exceeds 1 GB. If it does, manually adjust or delete the row.

3.8 Error Message "connection failure error" Is Displayed During GDS Data Transmission

Symptom

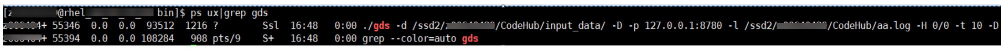
Error Message "connection failure error" is displayed during GDS data transmission.

Possible Causes

1. The GDS process breaks down. Run the following command to check whether the GDS process breaks down:

   ```
   ps ux|grep gds
   ```

   If the following information is displayed, the GDS process is started successfully:

   ```
   [[redacted]@rhel[redacted] bin]$ ps ux|grep gds
   [redacted]+ 55346  0.0  0.0  93512  1216 ?        Ssl  16:48   0:00 ./gds -d /ssd2/[redacted]/CodeHub/input_data/ -D -p 127.0.0.1:8780 -l /ssd2/[redacted]/CodeHub/aa.log -H 0/0 -t 10 -D
   [redacted]+ 55394  0.0  0.0 108284   908 pts/9    S+   16:48   0:00 grep --color=auto gds
   ```

2. The GDS startup parameter **-H** is incorrectly configured.

   -H *address_string*: indicates servers that are allowed to connect to and use GDS. The value must be in CIDR format. Set this parameter to allow the GaussDB(DWS) cluster to use GDS to import data. Ensure that the configured network segment covers all servers in the GaussDB(DWS) cluster.

Handling Procedure

1. Restart GDS. For details, see **Installing, Configuring, and Starting GDS**.
2. Change the parameter **–H** in the GDS startup command to **0/0** to check whether the fault is caused by this parameter. If the commands can be executed after the parameter value is changed to **0/0**, the original parameter settings are improper and the configured network segment does not contain all servers in the GaussDB(DWS) cluster.

3.9 Data to Be Imported Contains Chinese When the DataArts Studio Service Is Used to Create a GaussDB(DWS) Foreign Table

Symptom

When a user uses the DataArts Studio service to create a GaussDB(DWS) foreign table and specify UTF-8 as the OBS file encoding format in the statement for creating the foreign table. However, an error occurs when the user imports data.

Possible Causes

The source file stored on OBS contains non-UTF-8 data.

Handling Procedure

Check whether the source file contains non-UTF-8 data, for example, Chinese characters. If the source file contains non-UTF-8 data, convert the source file into the UTF-8 format, upload the converted file to OBS again, and import the data.

4 Database Parameter Modification

4.1 How to Change a Database's Default Time Zone When the Database Time Is Different from the System Time

Symptom

The database time is inconsistent with the operating system time. After a user queries the default database time **SYSDATE**, it is found that the database time is eight hours later than the Beijing time. As a result, the updated data cannot be accurately located.

Possible Causes

The UTC time zone is used to display and interpret GaussDB(DWS) database timestamps. If the operating system's time zone is not the UTC, time of the GaussDB(DWS) database will be inconsistent with the system time. Generally, the time zone of the cluster does not need to be changed. Configuring the time zone of the client may affect SQL execution.

Prerequisites

You are advised to modify the **timezone** parameter during off-peak hours.

Handling Procedure

Method 1: Change the default time zone of the database in a GaussDB(DWS) cluster.

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

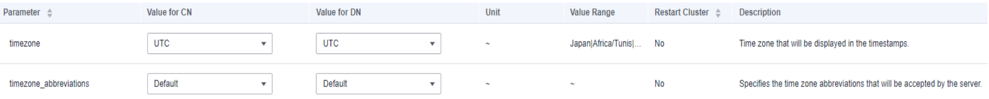
Step 2 In the navigation tree on the left, click **Clusters**.

Step 3 In the cluster list, find the target cluster and click its name. The **Basic Information** page is displayed.

Step 4 Click the **Parameter Modifications** tab and change the value of parameter **timezone** to your time zone. Then click **Save**.

Step 5 In the **Modification Preview** dialog box, confirm the modification and click **Save**.

Step 6 (Optional) Check the **Restart Cluster** column of the **timezone** parameter. It indicates whether you need to restart the cluster to make the parameter modification take effect.



Parameter	Value for CN	Value for DN	Unit	Value Range	Restart Cluster	Description
timezone	UTC	UTC	~	Japan|Africa/Tunis|...	No	Time zone that will be displayed in the timestamps.
timezone_abbreviations	Default	Default	~	~	No	Specifies the time zone abbreviations that will be accepted by the server.

NOTE

The modification of the **timezone** parameter takes effect immediately. You do not need to restart the cluster.

----End

Method 2: Run background commands to query and change the database time zone.

Step 1 Query the time zone and current time of the client. The time zone of the client is UTC, and the now() function returns the current time.

```
show time zone;
 TimeZone
----------
 UTC
(1 row)

select now();
              now
-------------------------------
 2022-05-16 06:05:58.711454+00
(1 row)
```

Step 2 Create a data table. **timestamp** and **timestamptz** are common time types. **timestamp** does not store the time zone, and **timestamptz** does.

```
CREATE TABLE timezone_test (id int, t1 timestamp, t2 timestamptz) DISTRIBUTE BY HASH (id);

\d timezone_test
           Table "public.timezone_test"
 Column |            Type             | Modifiers
--------+-----------------------------+-----------
 id     | integer                     |
 t1     | timestamp without time zone |
 t2     | timestamp with time zone    |
```

Step 3 Insert the current time into the **timezone_test** table and query the current table.

```
insert into timezone_test values (1, now(), now() );
show time zone;
 TimeZone
----------
 UTC
(1 row)
select * from timezone_test;
 id |             t1             |              t2
----+----------------------------+-------------------------------
  1 | 2022-05-16 06:10:04.564599 | 2022-05-16 06:10:04.564599+00
(1 row)
```

The **t1** (**timestamp** type) parameter discards the time zone information when saving data. The **t2** (**timestamptz** type) parameter saves the time zone information.

Step 4 Set the time zone of the client to UTC-8. Query the **timezone_test** table again.

```
set time zone 'UTC-8';
show time zone;
 TimeZone
----------
 UTC-8
(1 row)
select now();
              now
-------------------------------
 2022-05-16 14:13:43.175416+08
(1 row)
```

Step 5 Insert the current time to the **timezone_test** table and query the table. The value inserted to **t1** is UTC-8 time, and **t2** converts the time based on the time zone of the client.

```
insert into timezone_test values (2, now(), now() );
select * from timezone_test;
 id |             t1             |              t2
----+----------------------------+-------------------------------
  1 | 2022-05-16 06:10:04.564599 | 2022-05-16 14:10:04.564599+08
  2 | 2022-05-16 14:15:03.715265 | 2022-05-16 14:15:03.715265+08
(2 rows)
```

NOTE

- The **timestamp** type is affected the time zone used when data is inserted. The query result is not affected by the time zone of the client.
- The **timestamptz** type records the time zone information used when data is inserted. In a query, the time is converted based on the time zone of the client.

----End

4.2 Error "Cannot get stream index, maybe comm_max_stream is not enough" Is Reported

Symptom

When a user executes a task, the following error message is displayed: "ERROR: Failed to connect dn_6001_6002, detail:1021 Cannot get stream index, maybe comm_max_stream is not enough."

Possible Causes

The **comm_max_datanode** parameter of the user's database is set to the default value **1024**. During batch processing, the number of streams between DNs is 600 to 700. If temporary queries are performed at the same time, the total streams may exceed the specified upper limit, incurring the said error.

Cause Analysis

1. The GUC parameter **comm_max_stream** indicates the maximum number of streams between any two DNs.

On the CN, run the following command to query the stream status between any two DNs on the CN:

```
SELECT node_name,remote_name,count(*) FROM pgxc_comm_send_stream group by 1,2 order by 3
desc limit 100;
```

On a DN, run the following command to query the stream status between the DN and other DNs:

```
SELECT node_name,remote_name,count(*) FROM pg_comm_send_stream group by 1,2 order by 3
desc limit 100;
```

2. The value of **comm_max_stream** must be greater than: Number of concurrent data streams x Number of operators in each stream x Square of SMP.

   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.

   - 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**.

Handling Procedure

According to the evaluation result, the memory is sufficient. Change the value of **comm_max_stream** to **2048**. (The value **2048** is only an example. Set the parameter as needed.)

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

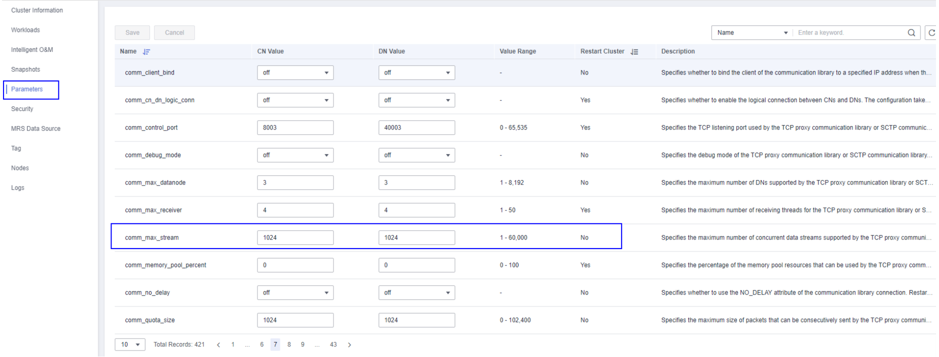
Step 2 In the navigation pane on the left, choose **Clusters**.

Step 3 In the cluster list, find the target cluster and click its name. The cluster information page is displayed.

Step 4 Choose **Parameters** tab and modify the value of parameter **comm_max_stream**. Click **Save**.

Step 5 In the **Modification Preview** dialog box, confirm the modification and click **Save**.

Step 6 If **No** is displayed in the **Restart Cluster** column of the **comm_max_stream** parameter, the parameter modification takes effect immediately without restart.



Name	CN Value	DN Value	Value Range	Restart Cluster	Description
comm_client_bind	off	off	-	No	Specifies whether to bind the client of the communication library to a specified IP address when th...
comm_cn_dn_logic_conn	off	off	-	Yes	Specifies whether to enable the logical connection between CNs and DNs. The configuration take...
comm_control_port	8003	40003	0 - 65,535	Yes	Specifies the TCP listening port used by the TCP proxy communication library or SCTP communic...
comm_debug_mode	off	off	-	No	Specifies the debug mode of the TCP proxy communication library or SCTP communication library...
comm_max_datanode	3	3	1 - 8,192	No	Specifies the maximum number of DNs supported by the TCP proxy communication library or SCT...
comm_max_receiver	4	4	1 - 50	Yes	Specifies the maximum number of receiving threads for the TCP proxy communication library or S...
comm_max_stream	1024	1024	1 - 60,000	No	Specifies the maximum number of concurrent data streams supported by the TCP proxy communi...
comm_memory_pool_percent	0	0	0 - 100	Yes	Specifies the percentage of the memory pool resources that can be used by the TCP proxy comm...
comm_no_delay	off	off	-	No	Specifies whether to use the NO_DELAY attribute of the communication library connection. Restar...
comm_quota_size	1024	1024	0 - 102,400	No	Specifies the maximum size of packets that can be consecutively sent by the TCP proxy communi...

----End

4.3 SQL Execution Fails With the Error Message "canceling statement due to statement timeout" Reported

Symptom

When an SQL statement is executed for more than 2 hours, the following error information is displayed:

```
ERROR: canceling statement due to statement timeoutTime.
```

Possible Causes

If the execution time of a statement exceeds the time specified by **statement_timeout**, an error is reported and the statement execution exits.

Handling Procedure

Method 1: Modify the **statement_timeout** parameter on the console.

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

Step 2 In the navigation pane on the left, choose **Clusters**.

Step 3 In the cluster list, find the target cluster and click its name. The cluster information page is displayed.

Step 4 Click the **Parameter Modifications** tab and modify the value of parameter session_timeout. Click **Save**.

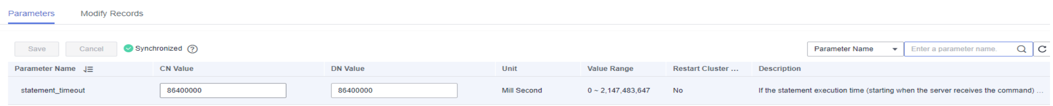
NOTE

By default, GaussDB(DWS) does not trigger SQL timeout. The default value of **statement_timeout** is **0**. If you have manually modified this parameter, you are advised to change it back to the default value **0** or set it to a proper value to prevent SQL timeout affecting other tasks.

Step 5 In the **Modification Preview** dialog box, confirm the modification and click **Save**.

Step 6 If **No** is displayed in the **Restart Cluster** column of the **statement_timeout** parameter, the parameter modification takes effect immediately without restart.

Figure 4-1 Modifying the **statement_timeout** parameter



Parameter Name	CN Value	DN Value	Unit	Value Range	Restart Cluster ...	Description
statement_timeout	86400000	86400000	Mill Second	0 ~ 2,147,483,647	No	If the statement execution time (starting when the server receives the command) ...

----End

Method 2: Connect to the cluster and run an SQL command to change the value of **statement_timeout**.

- Use the **SET** statement to change the value (session level):
  ```
  SET statement_timeout TO 0;
  ```
- Run the **ALTER ROLE** statement to change the value (user level):
  ```
  ALTER USER username SET statement_timeout TO 600000;
  ```
  In the preceding command, **username** indicates the user name of the database for which the SQL statement timeout interval is to be set.

5 Account/Permission/Password

5.1 How Do I Unlock an Account?

Symptom

The account has been locked is displayed when an account attempts to access a cluster.

Possible Causes

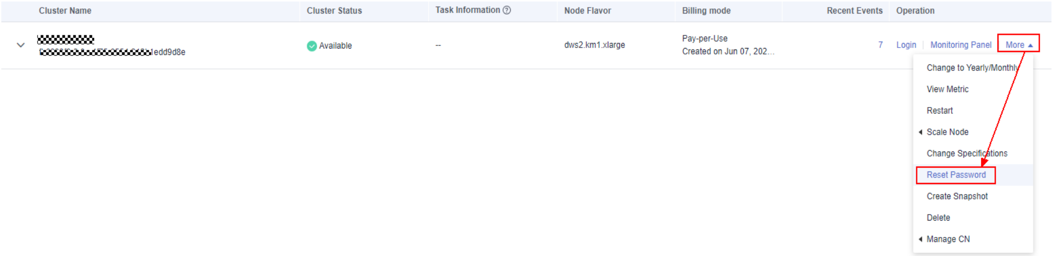
When you connect to a database in a cluster, if the number of consecutive incorrect password attempts reaches the upper limit, the account will be locked. The number of incorrect password attempts is specified by the GUC parameter **failed_login_attempts**, and the default value is 10.

NOTE

You can view audit logs to locate the cause of account locking. For details, see **Account Still Locked After Password Resetting**.

Method for Unlocking the Administrator (dbadmin by Default)

You can log in to the GaussDB(DWS) management console to reset the administrator password. When the password is reset, the account is automatically unlocked. On the console, go to the **Clusters** page, locate the required cluster, and choose **More > Reset Password**.



Cluster Name	Cluster Status	Task Information	Node Flavor	Billing mode	Recent Events	Operation
[redacted] [redacted]edd9d8e	Available	--	dws2.km1.xlarge	Pay-per-Use Created on Jun 07, 202...	7	Login Monitoring Panel More

Method for Unlocking Common Database Users

Connect to the database as the administrator (**dbadmin** by default) and run the following command to unlock a database user (replace **user_name** with the name of the locked user).

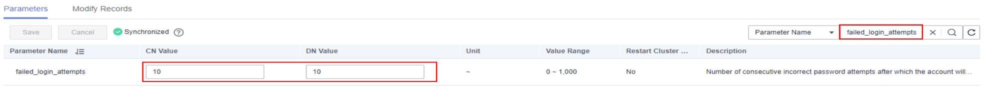
```
gsql -d gaussdb -p 8000 -U dbadmin -W Password -h Cluster IP address
ALTER USER user_name ACCOUNT UNLOCK;
```

Setting the Number of Times of Failed Login

You can set the maximum number of incorrect password attempts by configuring the **failed_login_attempts** parameter on the **Parameter Modifications** tab of the cluster. When **failed_login_attempts** is set to **0**, the number of incorrect password attempts is unlimited. You are not advised to set **failed_login_attempts** to **0**.

Perform the following steps:

1. Log in to the GaussDB(DWS) management console.
2. In the navigation tree on the left, click **Clusters**.
3. In the cluster list, find the target cluster and click the cluster name. The **Basic Information** page is displayed.
4. Enter the **Parameter Modifications** tab page, locate the **failed_login_attempts** parameter, change its value, and click **Save**. After confirming the modification, click **Save**.



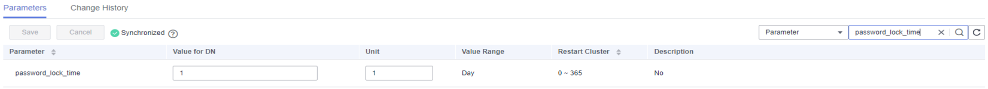
Parameter Name	CN Value	DN Value	Unit	Value Range	Restart Cluster ...	Description
failed_login_attempts	10	10	~	0 ~ 1,000	No	Number of consecutive incorrect password attempts after which the account will...

Setting the Time for Automatically Unlocking a Locked Account

After an account is locked, you can set the **password_lock_time** parameter to specify the automatic unlocking time. When the locking time exceeds the value of **password_lock_time**, the account is automatically unlocked. 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.

Perform the following steps:

1. Log in to the GaussDB(DWS) management console.
2. In the navigation tree on the left, click **Clusters**.
3. In the cluster list, find the target cluster and click the cluster name. The **Basic Information** page is displayed.
4. Enter the **Parameter Modifications** tab page, locate the **password_lock_time** parameter, change its value, and click **Save**. After confirming the modification, click **Save**.



Parameter	Value for DN	Unit	Value Range	Restart Cluster	Description
password_lock_time	1	1	Day	0 ~ 365	No

5.2 Account Still Locked After Password Resetting

Symptom

When a user connects to the cluster, the system displays a message indicating that the user is locked. After the user password is reset and the customer logs in again, the system still displays the message.

```
FATAL: The account has been locked.
```

Possible Causes

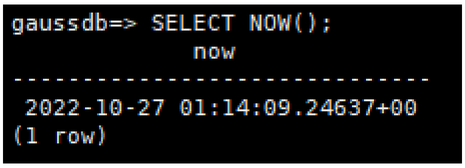
By default, a user will be locked if the user enters incorrect passwords for 10 consecutive times. The maximum number of incorrect password attempts is specified by the **failed_login_attempts** parameter. To modify the parameter, see **Setting the Number of Times of Failed Login**.

After the password is reset, the user is still locked. This may be caused by another user or application that has made 10 times of incorrect password attempts after the password is reset.

Handling Procedure

Step 1 Connect to the database as the system administrator **dbadmin** and run the following SQL statement to check the system time:

```
SELECT now();
```



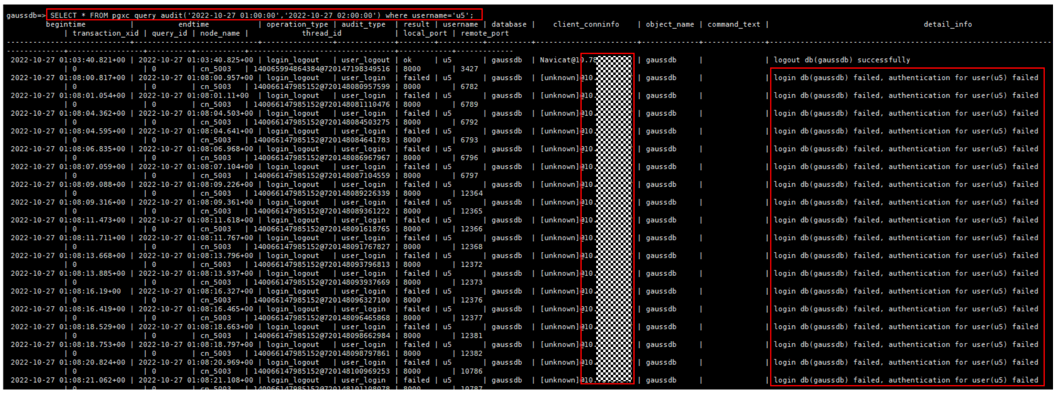
```
gaussdb=> SELECT NOW();
              now
-------------------------------
 2022-10-27 01:14:09.24637+00
(1 row)
```

The command output shows that the default system time on GaussDB(DWS) is the UTC time.

Step 2 Run the following SQL statement to query the client connection: In the preceding command:

- **username** should be replaced with the name of the locked user.
- The time period should be changed base actual requirements. For example, if you want to query the connection status from 09:00 to 10:00 (Beijing time), you need to convert the Beijing time to the UTC time, which is 01:00 to 02:00.

```
SELECT * FROM pgxc_query_audit('2022-10-27 01:00:00','2022-10-27 02:00:00') where
username='username';
```



```
gaussdb=> SELECT * FROM pgxc_query_audit('2022-10-27 01:00:00','2022-10-27 02:00:00') where username='u5';
         begintime          |          endtime           | operation_type | audit_type  | result | username | database |    client_conninfo    | object_name | command_text |                           detail_info
            | transaction_xid | query_id | node_name |            thread_id             | local_port | remote_port
----------------------------+----------------------------+----------------+-------------+--------+----------+----------+-----------------------+-------------+--------------+-----------------------------------------------------------------
------------+-----------------+----------+-----------+----------------------------------+------------+------------
 2022-10-27 01:03:40.821+00 | 2022-10-27 01:03:40.825+00 | login_logout   | user_logout | ok     | u5       | gaussdb  | Navicat@10.78[redacted] | gaussdb     |              | logout db(gaussdb) successfully
            | 0               | 0        | cn_5003   | 140065994864384@720147198349516 | 8000       | 3427
 2022-10-27 01:08:00.817+00 | 2022-10-27 01:08:00.957+00 | login_logout   | user_login  | failed | u5       | gaussdb  | [unknown]@10.[redacted] | gaussdb     |              | login db(gaussdb) failed, authentication for user(u5) failed
            | 0               | 0        | cn_5003   | 140066147985152@720148080957599 | 8000       | 6782
 2022-10-27 01:08:01.054+00 | 2022-10-27 01:08:01.11+00  | login_logout   | user_login  | failed | u5       | gaussdb  | [unknown]@10.[redacted] | gaussdb     |              | login db(gaussdb) failed, authentication for user(u5) failed
            | 0               | 0        | cn_5003   | 140066147985152@720148081110476 | 8000       | 6789
 2022-10-27 01:08:04.362+00 | 2022-10-27 01:08:04.503+00 | login_logout   | user_login  | failed | u5       | gaussdb  | [unknown]@10.[redacted] | gaussdb     |              | login db(gaussdb) failed, authentication for user(u5) failed
            | 0               | 0        | cn_5003   | 140066147985152@720148084503275 | 8000       | 6792
 2022-10-27 01:08:04.595+00 | 2022-10-27 01:08:04.641+00 | login_logout   | user_login  | failed | u5       | gaussdb  | [unknown]@10.[redacted] | gaussdb     |              | login db(gaussdb) failed, authentication for user(u5) failed
            | 0               | 0        | cn_5003   | 140066147985152@720148084641783 | 8000       | 6793
 2022-10-27 01:08:06.835+00 | 2022-10-27 01:08:06.968+00 | login_logout   | user_login  | failed | u5       | gaussdb  | [unknown]@10.[redacted] | gaussdb     |              | login db(gaussdb) failed, authentication for user(u5) failed
            | 0               | 0        | cn_5003   | 140066147985152@720148086967967 | 8000       | 6796
 2022-10-27 01:08:07.059+00 | 2022-10-27 01:08:07.104+00 | login_logout   | user_login  | failed | u5       | gaussdb  | [unknown]@10.[redacted] | gaussdb     |              | login db(gaussdb) failed, authentication for user(u5) failed
            | 0               | 0        | cn_5003   | 140066147985152@720148087104559 | 8000       | 6797
 2022-10-27 01:08:09.088+00 | 2022-10-27 01:08:09.226+00 | login_logout   | user_login  | failed | u5       | gaussdb  | [unknown]@10.[redacted] | gaussdb     |              | login db(gaussdb) failed, authentication for user(u5) failed
            | 0               | 0        | cn_5003   | 140066147985152@720148089226339 | 8000       | 12364
 2022-10-27 01:08:09.316+00 | 2022-10-27 01:08:09.361+00 | login_logout   | user_login  | failed | u5       | gaussdb  | [unknown]@10.[redacted] | gaussdb     |              | login db(gaussdb) failed, authentication for user(u5) failed
            | 0               | 0        | cn_5003   | 140066147985152@720148089361222 | 8000       | 12365
 2022-10-27 01:08:11.473+00 | 2022-10-27 01:08:11.618+00 | login_logout   | user_login  | failed | u5       | gaussdb  | [unknown]@10.[redacted] | gaussdb     |              | login db(gaussdb) failed, authentication for user(u5) failed
            | 0               | 0        | cn_5003   | 140066147985152@720148091618765 | 8000       | 12366
 2022-10-27 01:08:11.711+00 | 2022-10-27 01:08:11.767+00 | login_logout   | user_login  | failed | u5       | gaussdb  | [unknown]@10.[redacted] | gaussdb     |              | login db(gaussdb) failed, authentication for user(u5) failed
            | 0               | 0        | cn_5003   | 140066147985152@720148091767827 | 8000       | 12368
 2022-10-27 01:08:13.668+00 | 2022-10-27 01:08:13.796+00 | login_logout   | user_login  | failed | u5       | gaussdb  | [unknown]@10.[redacted] | gaussdb     |              | login db(gaussdb) failed, authentication for user(u5) failed
            | 0               | 0        | cn_5003   | 140066147985152@720148093796813 | 8000       | 12372
 2022-10-27 01:08:13.885+00 | 2022-10-27 01:08:13.937+00 | login_logout   | user_login  | failed | u5       | gaussdb  | [unknown]@10.[redacted] | gaussdb     |              | login db(gaussdb) failed, authentication for user(u5) failed
            | 0               | 0        | cn_5003   | 140066147985152@720148093937669 | 8000       | 12373
 2022-10-27 01:08:16.19+00  | 2022-10-27 01:08:16.327+00 | login_logout   | user_login  | failed | u5       | gaussdb  | [unknown]@10.[redacted] | gaussdb     |              | login db(gaussdb) failed, authentication for user(u5) failed
            | 0               | 0        | cn_5003   | 140066147985152@720148096327100 | 8000       | 12376
 2022-10-27 01:08:16.419+00 | 2022-10-27 01:08:16.465+00 | login_logout   | user_login  | failed | u5       | gaussdb  | [unknown]@10.[redacted] | gaussdb     |              | login db(gaussdb) failed, authentication for user(u5) failed
            | 0               | 0        | cn_5003   | 140066147985152@720148096465868 | 8000       | 12377
 2022-10-27 01:08:18.529+00 | 2022-10-27 01:08:18.663+00 | login_logout   | user_login  | failed | u5       | gaussdb  | [unknown]@10.[redacted] | gaussdb     |              | login db(gaussdb) failed, authentication for user(u5) failed
            | 0               | 0        | cn_5003   | 140066147985152@720148098662984 | 8000       | 12381
 2022-10-27 01:08:18.753+00 | 2022-10-27 01:08:18.797+00 | login_logout   | user_login  | failed | u5       | gaussdb  | [unknown]@10.[redacted] | gaussdb     |              | login db(gaussdb) failed, authentication for user(u5) failed
            | 0               | 0        | cn_5003   | 140066147985152@720148098797861 | 8000       | 12382
 2022-10-27 01:08:20.824+00 | 2022-10-27 01:08:20.969+00 | login_logout   | user_login  | failed | u5       | gaussdb  | [unknown]@10.[redacted] | gaussdb     |              | login db(gaussdb) failed, authentication for user(u5) failed
            | 0               | 0        | cn_5003   | 140066147985152@720148100969253 | 8000       | 10786
 2022-10-27 01:08:21.062+00 | 2022-10-27 01:08:21.108+00 | login_logout   | user_login  | failed | u5       | gaussdb  | [unknown]@10.[redacted] | gaussdb     |              | login db(gaussdb) failed, authentication for user(u5) failed
            | 0               | 0        | cn_5003   | 140066147985152@720148101108078 | 8000       | 10787
```

The preceding command output shows that the client whose IP address is x.x.x.x has made many attempts for connection using incorrect passwords.

Step 3 Perform either of the following operations based on the actual service situation:

- If the IP address obtained in step 2 belongs to a job, stop the job connection, connect to the database as the system administrator **dbadmin**, run the following SQL statement to unlock the user, then configure the job with the correct password.

```
ALTER USER username ACCOUNT UNLOCK;
```

- If you are not sure which job the IP address belongs to, change the value of **failed_login_attempts** to **0** by referring to **Setting the Number of Times of Failed Login**, and then run the following SQL statement to reset a new password. In this way, incorrect password attempts will no longer cause the account to be locked.

```
ALTER USER username IDENTIFIED BY '{Password}';
```

NOTICE

Setting the value of **failed_login_attempts** to **0** is only a temporary solution. To ensure database security, you are advised not to set **failed_login_attempts** to **0**. After locating the job and changing incorrect password, you are advised to set **failed_login_attempts** to 10.

----End

5.3 After the Permission for Querying Tables in a Schema Is Granted to a User, the User Still Cannot Query the Tables

Symptom

After an authorized user runs the **GRANT SELECT ON all tables in schema *schema_name* to *u1*** command to grant the access permission of tables in a schema to user **u1**, user **u1** still cannot access the tables.

Possible Causes

To authorize a user to access table or view objects in a schema, you also need to grant the **USAGE** permission of the schema to the user. Without this permission, the user can only view the names of the objects but cannot access them.

If you want to grant user **u1** the permission on tables to be created in the schema, run the **ALTER DEFAULT PRIVILEGES** command to change the default permission.

Handling Procedure

Log in to the database as a user with the schema permission and run the following command to grant the table permission in the schema to a specified user:

```
GRANT USAGE ON SCHEMA schema_name TO u1;
GRANT SELECT ON ALL TABLES IN SCHEMA schema_name TO u1;
```

Run the following command to grant the permission on the tables to be created in the schema to a specified user:

```
ALTER DEFAULT PRIVILEGES IN SCHEMA schema_name GRANT SELECT ON TABLES TO u1;
```

In the preceding SQL statements, **GRANT SELECT** indicates that the table query permission is assigned. If you want to assign other permissions to other users, see the **GRANT** syntax description.

NOTE

To grant the permission to query all tables in all schemas in the database to a user, query all the schemas in the **PG_NAMESPACE** system catalog then grant the permission to the user. Example:

```
SELECT 'grant select on all tables in '|| nspname || 'to u1' FROM pg_namespace;
```

5.4 How Do I Revoke the Permission of a User If grant select on table t1 to public Has Been Executed on a Table

Symptom

Assume that there are two common users **user1** and **user2**, and there are two tables **t1** and **t2** in the database. Run the following statement:

```
GRANT SELECT ON table t1 TO public;
```

user1 and **user2** have the permission to access table **t1**. After **user3** is created, **user3** also has the permission to access table **t1**. Running the **REVOKE SELECT on table t1 FROM user3;** statement to revoke **user3**'s permission to query table **t1** does not take effect.

```
testdb=# REVOKE SELECT ON table t1 FROM user3;
REVOKE
testdb=# \c - user3
Password for user user3:
Non-SSL connection (SSL connection is recommended when requiring high-security)
You are now connected to database "testdb" as user "user3".
testdb=> SELECT * FROM t1;
```

```
 a
---
(0 rows)

test=> SELECT relname, relacl FROM pg_class WHERE relname = 't1';
 relname |                     relacl
---------+-----------------------------------------------
 t1      | {user3=arwdDxt/user3,=r/user3}
(1 row)
```

Possible Causes

In the preceding problem, the revocation of **user3**'s permission to access table **t1** does not take effect because the **GRANT SELECT ON table t1 TO public;** statement has been executed. The keyword **public** in the statement indicates that the permission is granted to all roles, including the roles created later. Therefore, **user3** has the permission to access the table. **public** can be regarded as an implicitly defined group that contains all roles.

Therefore, after **REVOKE SELECT ON table t1 FROM user3;** is executed, although **user3** does not have the permission to access table **t1** (you can view the permissions of table **t1** in the **relacl** column in the **pg_class** system catalog), **user3** still has the **public** permission. Therefore, **user3** can still access the table.

Handling Procedure

You need to revoke the **public** permission of **user3** and then separately manage and control its permission. However, after the **public** permission is revoked, users (**user1** and **user2**) who could access the table may fail to do so, affecting services on the live network. Therefore, you need to run the **grant** command to grant the corresponding permissions to these users before revoking the **public** permission.

Step 1 View all users.

```
SELECT * FROM pg_user WHERE usesysid >= 16384;
 usename | usesysid | usecreatedb | usesuper | usecatupd | userepl |  passwd  | valbegin | valuntil |
respool    | parent | spacelimit | useconfig | nodegroup | tempspacelimit | spillspacelimit
---------+----------+-------------+----------+-----------+---------+----------+----------+----------+--------------
+--------+------------+-----------+-----------+----------------+-----------------
 jack    |    16408 | f           | f        | f         | f       | ******** |          |          | default_pool |      0 |
|            |           |                |
 tom     |    16412 | f           | f        | f         | f       | ******** |          |          | default_pool |      0 |
|            |           |                |
 user1   |    16437 | f           | f        | f         | f       | ******** |          |          | default_pool |      0 |
|            |           |                |
 user2   |    16441 | f           | f        | f         | f       | ******** |          |          | default_pool |      0 |
|            |           |                |
 user3   |    16448 | f           | f        | f         | f       | ******** |          |          | default_pool |      0 |
|            |           |                |
(5 rows)
```

Step 2 Run the **GRANT** statement to grant permissions to the original user.

```
GRANT select on table t1 TO jack,tom,user1,user2;
GRANT
```

Step 3 Revoke the **public** permission on the sample table **t1**.

```
REVOKE select on table t1 FROM public;
```

Step 4 Switch to **user3** and query sample table **t1**. The result shows that **user3**'s permission to access table **t1** has been revoked successfully.

```
testdb=# \c - user3
Password for user user3:
```

```
Non-SSL connection (SSL connection is recommended when requiring high-security)
You are now connected to database "testdb" as user "user3".
testdb=> SELECT * FROM t1;
ERROR:  permission denied for relation t1
```

----End

5.5 An Error Message Is Displayed When a Common User Executes the Statement for Creating or Deleting a GDS or OBS Foreign Table, Indicating that the User Does Not Have the Permission or the Permission Is Insufficient

Symptom

An administrator can execute the statement for creating a GDS or OBS foreign table, but an error "ERROR: permission denied to create foreign table in security mode" is reported when a common user executes the statement.

```
CREATE USER u1 PASSWORD '{password}';
SET current_schema = u;
CREATE FOREIGN TABLE customer_ft
(
    c_customer_sk             integer               ,
    c_customer_id             char(16)              ,
    c_current_cdemo_sk        integer               ,
    c_current_hdemo_sk        integer               ,
    c_current_addr_sk         integer
)
    SERVER gsmpp_server
    OPTIONS
(
    location 'gsfs://192.168.0.90:5000/customer1*.dat',
    FORMAT 'TEXT' ,
    DELIMITER '|',
    encoding 'utf8',
    mode 'Normal')
READ ONLY;
ERROR:  permission denied to create foreign table in security mode
```

Possible Causes

The error message indicates that the common user does not have the permission for creating a foreign table.

Query the user permissions in the case.

```
SELECT rolname,roluseft FROM pg_roles WHERE rolname ='u1' ORDER BY rolname desc;
 rolname | roluseft
---------+----------
 u1      | f
(1 row)
```

Handling Procedure

You can use the **ALTER USER** or **ALTER ROLE** syntax to specify the **USEFT** parameter, granting a role or user the permission to use foreign tables.

USEFT | NOUSEFT determines whether a new role or user can perform operations on foreign tables, such as creating, deleting, modifying, and reading/witting foreign tables.

- If **USEFT** is specified, the role or user can perform operations on foreign tables.
- The default value is **NOUSEFT**, indicating that the new role or user does not have permissions to perform operations on foreign tables.

To grant the permission to use foreign tables to a common user or role, run the following command as a database administrator:

```
ALTER USER user_name USEFT;
```

For details about how to modify user or role permissions, see **ALTER USER** or **ALTER ROLE**.

Common users or roles can create foreign tables after being granted the permission to use foreign tables.

5.6 After the all Permission Is Granted to the Schema of a User, the Error Message "ERROR: current user does not have privilege to role tom" Persists During Table Creation

Symptom

Assume that there are two users, **tom** and **jerry**. **jerry** wants to create a table in the schema with the same name as that of **tom**. **tom** grants the all permission of the schema to **jerry**. However, an error is still reported when the table is created.

```
dbtest=# GRANT all on schema tom to jerry;
GRANT
dbtest=# \c - jerry
Password for user jerry:
Non-SSL connection (SSL connection is recommended when requiring high-security)
You are now connected to database "dbtest" as user "jerry".
dbtest=>
dbtest=> CREATE TABLE tom.t(a int);
ERROR:  current user does not have privilege to role tom
```

Possible Causes

According to the error message, **jerry** requires the permission of the role **tom**.

Handling Procedure

After the permission of the role **tom** is granted to **jerry**, the table is created successfully.

```
dbtest=# GRANT tom to jerry;
GRANT ROLE
dbtest=# \c - jerry
Password for user jerry:
Non-SSL connection (SSL connection is recommended when requiring high-security)
You are now connected to database "dbtest" as user "jerry".
```

```
dbtest=>
dbtest=> CREATE TABLE tom.t(a int);
NOTICE:  The 'DISTRIBUTE BY' clause is not specified. Using 'a' as the distribution column by default.
HINT:  Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
```

5.7 An Error Message Is Reported During Statement Execution, Indicating that the User Does Not Have the Required Permission

Symptom

The following error message is displayed after statement execution.

```
ERROR:  permission denied for xxx
```

Possible Causes

The user does not have the corresponding permission and cannot access or perform operations on the table or schema.

Handling Procedure

Step 1 Grant permissions to tables or schemas using **GRANT**. If you want user **jerry** to have the query permission on all tables created by **tom** and the tables to be created, perform the following operations:

- Grant the schema permissions of user **tom** to user **jerry**.

  ```
  GRANT USAGE ON SCHEMA tom TO jerry;
  ```

- Grant the **SELECT** permission on the tables created by user **tom** to user **jerry**.

  ```
  GRANT SELECT ON ALL TABLES IN SCHEMA tom TO jerry;
  ```

- Grant the **SELECT** permission on the tables created by user **tom** in the schema with the same name to user **jerry**.

  ```
  ALTER DEFAULT PRIVILEGES FOR USER tom IN SCHEMA tom GRANT SELECT ON TABLE TO jerry;
  ```

----End

5.8 Failed to Run the create extension Command and An Error Indicating No Permission Is Reported

Symptom

The **create extension** command failed to be run and the following error message was displayed:

```
ERROR: permission denied to create extension "uuid-ossp" Hint: Must be superuser to create this extension.
```

Possible Causes

GaussDB(DWS) does not support the extension feature of the PostgreSQL community.

5.9 A User Cannot Be Deleted Due to Its Dependencies

Symptom

When a user is no longer used or the role of the user changes, the user account needs to be deregistered and the permission needs to be revoked. However, when the user is deleted, an error message similar to **role "u1" cannot be dropped because some objects depend on it** is displayed.

For example, if you want to delete user **u1**, the following information is displayed:

```
testdb=# DROP USER u1;
ERROR:  role "u1" cannot be dropped because some objects depend on it
DETAIL:  owner of database testdb
3 objects in database gaussdb
```

Possible Causes

If the permissions of a user or role are complex and have many dependencies, an error message is displayed when you delete the user or role, indicating that the user or role has dependencies and cannot be deleted. Obtain the following information based on the error information:

- The user to be deleted is the owner of database **testdb**.
- It has three dependent objects are in the GaussDB database.

Handling Procedure

- **If a user to be deleted is the owner of a database. You need to reassign the object ownerships to another user.** Use either of the following methods:

  Method 1: Transfer the database ownership to another user. For example, run the **ALTER** statement to change the owner user **u1** of the **testdb** database to **u2**.

  ```
  testdb=# ALTER DATABASE testdb OWNER to u2;
  ALTER DATABASE
  testdb=# \l
                                List of databases
     Name    |  Owner  | Encoding |   Collate    |    Ctype    |    Access privileges
  -----------+----------+----------+-------------+-------------+-------------------------
   testdb    | u2       | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
  (4 rows)
  ```

  After the command for deleting the **u1** user is executed, the message "**owner of database testdb**" is not displayed.

  ```
  testdb=# DROP USER u1;
  ERROR:  role "u1" cannot be dropped because some objects depend on it
  DETAIL:  3 objects in database gaussdb
  ```

  Method 2: If the **testdb** database is no longer required, delete it. Change the owners of all database objects owned by **u1** to **u2**.

  ```
  testdb=# REASSIGN OWNED BY u1 TO u2;
  REASSIGN OWNED
  ```

  Clear objects whose owner is **u1**. Exercise caution when running this command. Schemas with the same name will also be deleted.

  ```
  testdb=# DROP OWNED by u1;
  DROP OWNED
  ```

- **If the user to be deleted has dependencies. You need to remove the dependencies before you delete it.** The method is as follows:
  a. Identify dependencies. According to the error information "3 objects in database gaussdb", three objects in the GaussDB database depend on **u1**. Due to the system catalog dependency in the database, detailed information about dependent objects is not printed in other databases, but is printed in the GaussDB database when **DROP USER** is executed in GaussDB database.

     Run the following command to connect to the GaussDB database:

     ```
     gaussdb=# DROP USER u1;
     ERROR:  role "u1" cannot be dropped because some objects depend on it
     DETAIL:  privileges for table pg_class
     privileges for schema u2
     ```

     The obtained dependency details are as follows:

     i. **privileges for table pg_class**: permissions of user **u1** on **pg_class**.

     ii. Permission of user **u1** on schema **u2**.

  b. Revoke the permissions on dependent objects.

     ```
     gaussdb=# SELECT relname,relacl FROM pg_class WHERE relname = 'pg_class';
      relname  |              relacl
     ----------+-----------------------------------
      pg_class | {=r/Ruby,u1=r/Ruby}
     (1 row)

     gaussdb=#SELECT nspname,nspacl FROM pg_namespace WHERE nspname = 'u2';
      nspname |           nspacl
     ---------+-----------------------------
      u2      | {u2=UC/u2,u2=LP/u2,u1=U/u2}

     gaussdb=# REVOKE SELECT ON TABLE pg_class FROM u1;
     REVOKE
     gaussdb=# REVOKE USAGE ON SCHEMA u2 FROM u1;
     REVOKE
     ```

  c. Delete the user again. The user can be deleted successfully, and the message indicating that dependencies exist is not displayed.

     ```
     gaussdb=# DROP USER u1;
     DROP USER
     ```

- **In some scenarios, the dependent objects of the user to be deleted are unknown, but the deletion still fails.** The following uses a constructed case to demonstrate how to handle this situation. Create user **u3** and assign the **SELECT** permission to user **u2**.

  ```
  testdb2=# DROP USER u3;
  ERROR:  role "u3" cannot be dropped because some objects depend on it
  DETAIL:  2 objects in database gaussdb
  ```

  a. The **pg_shdepend** system catalog records the OIDs of dependent objects and their dependencies. Obtain the OID of the user, and then search the system catalogs for the corresponding dependency records.

     ```
     testdb2=# SELECT oid ,rolname FROM pg_roles WHERE rolname = 'u3';
         oid     | rolname
     ------------+---------
      2147484573 | u3
     (1 row)

     gaussdb=# SELECT * FROM pg_shdepend WHERE refobjid = 2147484573;
      dbid  | classid |   objid    | objsubid | refclassid |  refobjid  | deptype | objfile
     -------+---------+------------+----------+------------+------------+---------+---------
      16073 |    2615 | 2147484575 |        0 |       1260 | 2147484573 | o       |
      16073 |    2615 | 2147484025 |        0 |       1260 | 2147484573 | a       |
     (2 rows)
     ```

The values of **dependType** may be different. There are two records. One indicates permission dependency (a), and the other indicates that the object is the owner of another object.

b. **classid** indicates the ID of the record table that records the object that depends on the user. You can use the **classid** to find the dependency in **pg_class**.

```
gaussdb=# SELECT relname,relacl FROM pg_class WHERE oid = 2615;
   relname    |     relacl
--------------+----------------
 pg_namespace | {=r/role23}
(1 row)
```

c. The query result shows that the record table is **pg_namespace**. It can be concluded that the object depends on the user is a schema. In **pg_namespace**, query the **objid** obtained in 1 to determine the specific object.

```
gaussdb=# SELECT nspname,nspacl FROM pg_namespace WHERE oid in
(2147484575,2147484025);
 nspname |                          nspacl
---------+----------------------------------------------------------
 u3      |
 u2      | {u2=UC/u2,u2=LP/u2,u3=U/u2}
(2 rows)
```

There are two schemas. **u3** is the schema with the same name as the user, and **u2** is the schema to which the permission is granted. Revoke the permission on the schema.

```
gaussdb=# REVOKE USAGE ON SCHEMA u2 FROM u3;
REVOKE
```

d. Delete user **u3**. The deletion is successful.

```
gaussdb=# DROP USER u3;
DROP USER
```

6 Cluster Performance

6.1 Lock Wait Detection

Scenario

In job development, locks in database transaction management generally refer to table-level locks. GaussDB(DWS) supports eight lock modes, ranging from 1 to 8 based on exclusive levels. Each lock mode conflicts with another lock mode. **Table 6-1** describes lock conflicts details.

Example: user **u1** holds the **RowExclusiveLock** lock when executing the **INSERT** transaction on table **test**. If user **u2** performs the **VACUUM FULL** transaction on the table **test**, a lock conflict occurs, and the lock of user **u2** will wait.

Common lock wait detection is performed by querying the **pgxc_lock_conflicts**, **pgxc_stat_activity**, **pgxc_thread_wait_status**, and **pg_locks** views. The **pgxc_lock_conflicts** view is supported in versions later than 8.1.x. The detection method varies depending on the cluster version.

Table 6-1 Lock conflicts

No.	Lock	Purpose	Conflict
1	AccessShareLock	SELECT	8
2	RowShareLock	SELECT FOR UPDATE/FOR SHARE	7 | 8
3	RowExclusiveLock	INSERT/UPDATE/DELETE	5 | 6 | 7 | 8
4	ShareUpdateExclusiveLock	VACUUM	4 | 5 | 6 | 7 | 8
5	ShareLock	CREATE INDEX	3 | 4 | 6 | 7 | 8

No.	Lock	Purpose	Conflict
6	ShareRowExclusiveLock	ROW SELECT...FOR UPDATE	3 | 4 | 5 | 6 | 7 | 8
7	ExclusiveLock	BLOCK ROW SHARE/ SELECT...FOR UPDATE	2 | 3 | 4 | 5 | 6 | 7 | 8
8	AccessExclusive-Lock	DROP CLASS/ VACUUM FULL	1 | 2 | 3 | 4 | 5 | 6 | 7 | 8

Procedure

Creating a lock wait:

Step 1 Open a new connection session, connect to the GaussDB(DWS) database as common user **u1**, and create a test table **u1.test** in SCHEMA **u1**.

```
CREATE TABLE test (id int, name varchar(50));
```

Step 2 Start transaction 1 and perform the INSERT operation.

```
START TRANSACTION;
INSERT INTO test VALUES (1, 'lily');
```

Step 3 Open a new connection session, connect to the GaussDB(DWS) database as the system administrator **dbadmin**, and perform the **VACUUM FULL** operation. The statement is blocked.

```
VACUUM FULL u1.test;
```

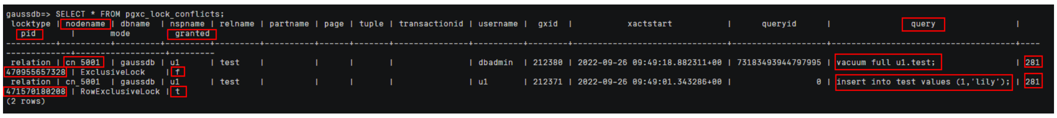
----End

Lock wait detection (8.1.x and later versions)

Step 1 Open a new connection session, connect to the GaussDB(DWS) database as the system administrator **dbadmin**, and check lock conflicts in the **pgxc_lock_conflicts** view.

As shown in the following figure, if the value of **granted** is **f**, the **VACUUM FULL** statement is waiting for another lock. If **granted** is **t**, the **INSERT** statement holds the lock. **nodename**: indicates the CN or DN where the lock is generated, for example, **cn_5001**.

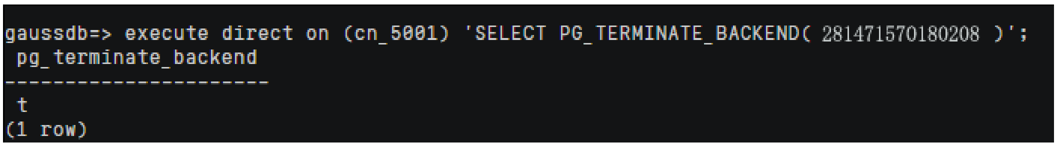
```
SELECT * FROM pgxc_lock_conflicts;
```



```
gaussdb=> SELECT * FROM pgxc_lock_conflicts;
 locktype | nodename | dbname  | nspname | relname | partname | page | tuple | transactionid | username |  gxid  |          xactstart            |      queryid      |              query                 |
  pid       |       mode        | granted
----------+----------+---------+---------+---------+----------+------+-------+---------------+----------+--------+-------------------------------+-------------------+------------------------------------+----
------------+-------------------+---------
 relation | cn_5001  | gaussdb | u1      | test    |          |      |       |               | dbadmin  | 212380 | 2022-09-26 09:49:18.882311+00 | 73183493944797995 | vacuum full u1.test;               | 281
470955657328 | ExclusiveLock     | f
 relation | cn_5001  | gaussdb | u1      | test    |          |      |       |               | u1       | 212371 | 2022-09-26 09:49:01.343286+00 |                 0 | insert into test values (1,'lily'); | 281
471570180208 | RowExclusiveLock  | t
(2 rows)
```

Step 2 Decide whether to terminate the lock based on the statement content. To terminate the lock, run the following statement: Obtain the value of *pid* from **Step 1**. In the preceding command, **cn_5001** indicates the **nodename** queried in the preceding step.

```
execute direct on (cn_5001) 'SELECT PG_TERMINATE_BACKEND(pid)';
```



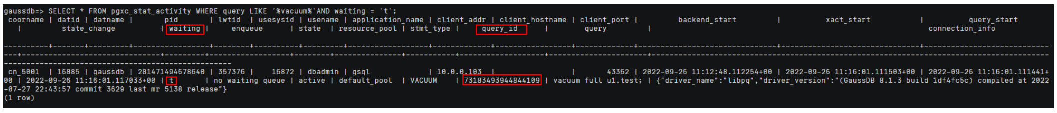
```
gaussdb=> execute direct on (cn_5001) 'SELECT PG_TERMINATE_BACKEND( 281471570180208 )';
 pg_terminate_backend
----------------------
 t
(1 row)
```

----End

Lock wait detection (8.0.x and earlier versions)

Step 1 Run the following statement in the database to obtain the value of **query_id** corresponding to the **VACUUM FULL** operation:

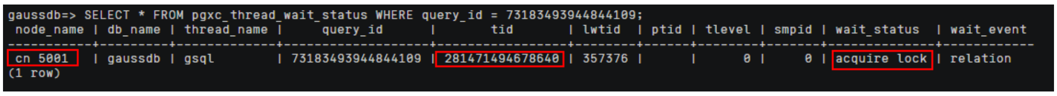
```
SELECT * FROM pgxc_stat_activity WHERE query LIKE '%vacuum%'AND waiting = 't';
```



```
gaussdb=> SELECT * FROM pgxc_stat_activity WHERE query LIKE '%vacuum%'AND waiting = 't';
 coorname | datid | datname |       pid       | lwtid  | usesysid | usename | application_name | client_addr | client_hostname | client_port |         backend_start         |          xact_start           |          query_start
   |        state_change        | waiting |     enqueue      | state  | resource_pool | stmt_type |     query_id      |        query        |                                   connection_info
----------+-------+---------+-----------------+--------+----------+---------+------------------+-------------+-----------------+-------------+-------------------------------+-------------------------------+------------------------------
---+----------------------------+---------+------------------+--------+---------------+-----------+-------------------+---------------------+--------------------------------------------------------------------------------------------
----------------------------------------------------
 cn_5001  | 16885 | gaussdb | 281471494678640 | 357376 |    16872 | dbadmin | gsql             | 10.0.0.103  |                 |       43362 | 2022-09-26 11:12:48.112254+00 | 2022-09-26 11:16:01.111503+00 | 2022-09-26 11:16:01.111441+
00 | 2022-09-26 11:16:01.117033+00 | t       | no waiting queue | active | default_pool  | VACUUM    | 73183493944844109 | vacuum full u1.test; | {"driver_name":"libpq","driver_version":"(GaussDB 8.1.3 build 1df4fc5c) compiled at 2022
-07-27 22:43:57 commit 3629 last mr 5138 release"}
(1 row)
```

Step 2 Run the following statement based on the obtained **query_id** to check whether lock wait exists and obtain the corresponding **tid**: In the preceding information, **query_id** is obtained from **Step 1**.

SELECT * FROM pgxc_thread_wait_status WHERE query_id = *{query_id}*;



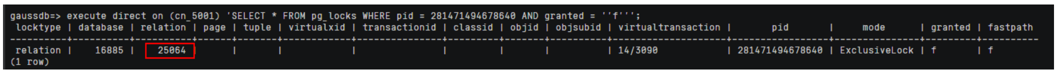
```
gaussdb=> SELECT * FROM pgxc_thread_wait_status WHERE query_id = 73183493944844109;
 node_name | db_name | thread_name |     query_id      |       tid       | lwtid  | ptid | tlevel | smpid | wait_status  | wait_event
-----------+---------+-------------+-------------------+-----------------+--------+------+--------+-------+--------------+------------
 cn_5001   | gaussdb | gsql        | 73183493944844109 | 281471494678640 | 357376 |      |      0 |     0 | acquire lock | relation
(1 row)
```

If **acquire lock** is displayed in **wait_status** in the command output, lock wait exists. Check the value of **node_name** and record it, for example, **cn_5001** or **dn_600x_600y**.

Step 3 Run the following statement to check the lock waited by the **VACUUM FULL** operation in **pg_locks**: The following uses **cn_5001** as an example. If the lock wait is on a DN, change it to the corresponding DN name. **pid** is obtained in **Step 2**.

Record the value of **relation** in the command output.

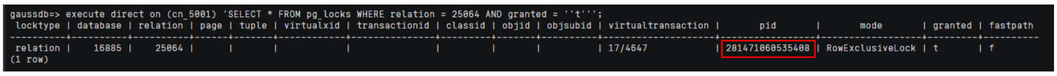
execute direct on *(cn_5001)* 'SELECT * FROM pg_locks WHERE pid = *{tid}* AND granted = ''f''';



```
gaussdb=> execute direct on (cn_5001) 'SELECT * FROM pg_locks WHERE pid = 281471494678640 AND granted = ''f''';
 locktype | database | relation | page | tuple | virtualxid | transactionid | classid | objid | objsubid | virtualtransaction |       pid       |     mode      | granted | fastpath
----------+----------+----------+------+-------+------------+---------------+---------+-------+----------+--------------------+-----------------+---------------+---------+----------
 relation |    16885 |    25064 |      |       |            |               |         |       |          | 14/3090            | 281471494678640 | ExclusiveLock | f       | f
(1 row)
```

Step 4 Query the **pg_locks** system catalog for the PID of the lock based on the **relation**, which is obtained in **Step 3**.

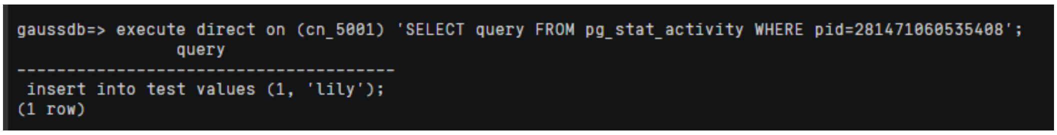
execute direct on *(cn_5001)* 'SELECT * FROM pg_locks WHERE relation = *{relation}* AND granted = ''t''';



```
gaussdb=> execute direct on (cn_5001) 'SELECT * FROM pg_locks WHERE relation = 25064 AND granted = ''t''';
 locktype | database | relation | page | tuple | virtualxid | transactionid | classid | objid | objsubid | virtualtransaction |       pid       |       mode       | granted | fastpath
----------+----------+----------+------+-------+------------+---------------+---------+-------+----------+--------------------+-----------------+------------------+---------+----------
 relation |    16885 |    25064 |      |       |            |               |         |       |          | 17/4647            | 281471060535408 | RowExclusiveLock | t       | f
(1 row)
```

Step 5 Run the following statement based on the **PID** to query the corresponding SQL statement: The value of **pid** is obtained in **Step 4**.

execute direct on *(cn_5001)* 'SELECT query FROM pg_stat_activity WHERE pid=*{pid}*';

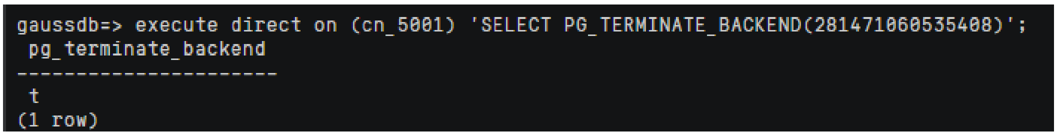


```
gaussdb=> execute direct on (cn_5001) 'SELECT query FROM pg_stat_activity WHERE pid=281471060535408';
                 query
----------------------------------------
 insert into test values (1, 'lily');
(1 row)
```

Step 6 Based on the statement content, determine whether to stop the statement or run **VACUUM FULL** again after the statement is complete. To terminate the lock, run the following statement: The value of **pid** is obtained in **Step 4**.

After the lock is terminated, run **VACUUM FULL** again.

```
execute direct on (cn_5001) 'SELECT PG_TERMINATE_BACKEND(pid)';
```



```
gaussdb=> execute direct on (cn_5001) 'SELECT PG_TERMINATE_BACKEND(281471060535408)';
 pg_terminate_backend
----------------------
 t
(1 row)
```

----End

6.2 During SQL Execution, a Table Deadlock Occurs and An Error Stating LOCK_WAIT_TIMEOUT Is Reported

Symptom

During SQL execution, lock wait timeout (LOCK_WAIT_TIMEOUT) is reported.

Possible Causes

Lock wait timeout is generally caused by the fact that another SQL statement has held the lock. The current SQL statement can be executed only after the SQL statement that holds the lock is successfully executed and releases the lock. If the lock wait time exceeds the specified value of the GUC parameter **lockwait_timeout**, the system reports the LOCK_WAIT_TIMEOUT error.

Handling Procedure

1. For clusters of 8.1.x or later, check lock conflicts in the **pgxc_lock_conflicts** view.

   ```
   SELECT * FROM pgxc_lock_conflicts;
   ```

   For clusters of 8.0.x and earlier versions, run the following SQL query to check whether there are blocked SQL statements. If yes, forcibly end the blocked SQL sessions.

   ```
   SELECT w.query as waiting_query,
   w.pid as w_pid,
   w.usename as w_user,
   l.query as locking_query,
   l.pid as l_pid,
   l.usename as l_user,
   n.nspname || '.' || c.relname as tablename
   from pg_stat_activity w join pg_locks l1 on w.pid = l1.pid
   and not l1.granted join pg_locks l2 on l1.relation = l2.relation
   and l2.granted join pg_stat_activity l on l2.pid = l.pid join pg_class c on c.oid = l1.relation join
   pg_namespace n on n.oid=c.relnamespace
   where w.waiting;
   ```

2. After the blocked table and the schema information are found, end the session based on the session ID.

   ```
   SELECT PG_TERMINATE_BACKEND(PID);
   ```

3. This fault is generally caused by improper service scheduling. It is recommended that the scheduling time of each service be properly arranged.

4. You can also set the GUC parameter **lockwait_timeout** to control the maximum wait time (wait timeout) of a single lock.

   The unit of **lockwait_timeout** is millisecond. The default value is 20 minutes.

   The **lockwait_timeout** parameter is of the SUSET type. Follow the instructions in **Configuring GUC Parameters** to configure the parameter.

6.3 Error "abort transaction due to concurrent update" Is Reported During SQL Execution

Symptom

The error message "abort transaction due to concurrent update" is reported indicating that lock waiting times out when SQL statements are executed.

Possible Causes

Concurrent operations from two transactions are executed on a single row in a table. As a result, the transaction that is operated later is rolled back.

For example:

Step 1 Open a connection session A, connect to the GaussDB(DWS) database as common user **u1**, create a test table **u1.test** in SCHEMA **u1**, and insert data into the table.

```
CREATE TABLE test (id int, name varchar(50));
INSERT INTO test VALUES (1, 'lily');
```

Step 2 Open a new connection session **session B**, start **transaction 1**, connect to the GaussDB(DWS) database as the system administrator **dbadmin**, and perform the **UPDATE** operation.

```
START TRANSACTION;

UPDATE u1.test SET id = 3 WHERE name = 'lily';
UPDATE 1
```

Step 3 Start **transaction 2** in **session A** and execute the same **UPDATE** statement. An error is reported.

```
START TRANSACTION;

UPDATE test SET id = 3 WHERE name = 'lily';
ERROR:  dn_6003_6004: abort transaction due to concurrent update test 289502.
```

----End

In the preceding case, two different transactions concurrently update the same record. There is no lock waiting. Instead, an error is reported: abort transaction due to concurrent update.

In practice, an error may be reported not only when the same record is concurrently updated. For other concurrent SQL operations such as **SELECT** and **DELETE**, the error "abort transaction due to concurrent update" may also be reported.

Handling Procedure

- Adjust the execution sequence of service logic and the SQL statements.

  Do not place the SQL statement that holds the lock for a long time in the front.

- Avoid large transactions.

  Split a large transaction into multiple small transactions for processing. Small transactions shorten the resource locking time and have a lower probability of conflicts.

- Control concurrency.

  Reduce the number of concurrent sessions as much as possible to reduce the probability of conflicts.

6.4 Solution to High Disk Usage and Cluster Read-Only

Checking the Disk Usage

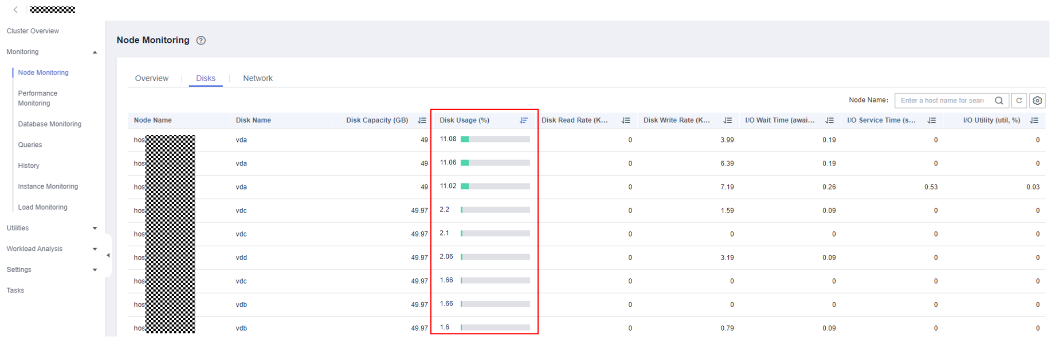
GaussDB(DWS) disk space is a high-value resource for users. It is closely related to cluster availability. Therefore, you need to pay close attention to the disk space and make response in a timely manner (in this section, disks refer to data disks).

To check the disk space, perform the following steps:

Step 1 Log in to the GaussDB(DWS) management console. Click **Clusters** on the left navigation pane. In the cluster list, click **Monitoring** on the right of the row that contains the desired cluster. The **Monitoring** page is displayed.

Step 2 Choose **Monitoring** > **Node Monitoring**. On the displayed page, choose the **Disks** tab. You can click ↓≡ on the right to sort the records based on disk usage.

To identify a data disk, check the disk capacity. If the disk capacity equals to the purchased capacity, the disk is a data disk.



Node Name	Disk Name	Disk Capacity (GB)	Disk Usage (%)	Disk Read Rate (K...	Disk Write Rate (K...	I/O Wait Time (awai...	I/O Service Time (s...	I/O Utility (util, %)
hos[redacted]	vda	49	11.08	0	3.99	0.19	0	0
hos[redacted]	vda	49	11.06	0	6.39	0.19	0	0
hos[redacted]	vda	49	11.02	0	7.19	0.26	0.53	0.03
hos[redacted]	vdc	49.97	2.2	0	1.59	0.09	0	0
hos[redacted]	vdc	49.97	2.1	0	0	0	0	0
hos[redacted]	vdd	49.97	2.06	0	3.19	0.09	0	0
hos[redacted]	vdc	49.97	1.66	0	0	0	0	0
hos[redacted]	vdb	49.97	1.66	0	0	0	0	0
hos[redacted]	vdb	49.97	1.6	0	0.79	0.09	0	0

----End

Fault Scenarios

- **Scenario 1: High Disk Usage**: The usage of all disks or more than half of the disks in the current cluster is greater than or equal to 70%.
- **Scenario 2: Disk Skew**: The difference between the highest usage and the lowest usage is greater than or equal to 10%.

- **Scenario 3: Cluster Read-only.** (The current read-only threshold is that the usage of a single data disk is greater than or equal to 90%.)

During routine processing, the database administrator can use the **Space Management and Control** to identify and block abnormal workloads to avoid the preceding scenarios.

NOTICE

1. Read-only is a self-protection mechanism of the GaussDB(DWS) system. It prevents GaussDB(DWS) instance startup failures caused by 100% disk usage.
2. In the preceding scenarios, the DMS sends alarm notifications.
3. In scenario 1, you can use the alarm subscription function to receive SMS messages or emails when the disk usage exceeds 70% or 75%. This allows you to clear data in advance. For details, see **Subscribing to Disk Space Alarms**.

Scenario 1: Clearing Data When the Disk Usage Is High

Periodically delete **dirty data** based on the query results. The method varies according to the cluster version.

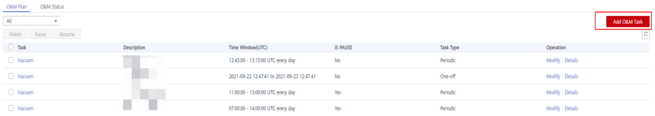
- **8.1.3 and later versions: Use the Intelligent O&M function on the management console to automatically clear dirty data.**

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

Step 2 Click the name of the target cluster.

Step 3 In the navigation pane, choose **Intelligent O&M**.

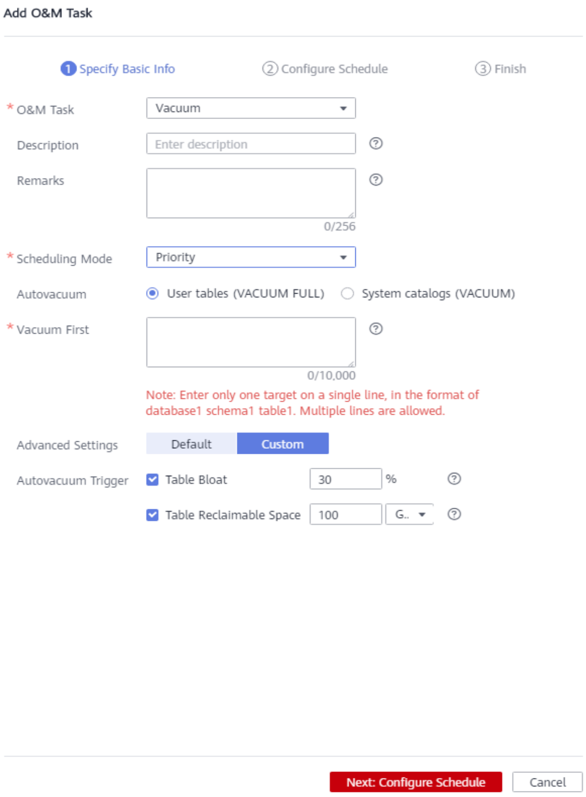
Step 4 Click the **O&M Plan** tab. Click **Add O&M Task**.



Task	Description	Time Window(UTC)	IS PAUSE	Task Type	Operation
Vacuum	[redacted]	12:45:00 - 13:15:00 UTC every day	No	Periodic	Modify | Details
Vacuum	[redacted]	2021-09-22 12:47:41 to 2021-09-23 12:47:41	No	One-off	Modify | Details
Vacuum	[redacted]	11:00:00 - 13:00:00 UTC every day	Yes	Periodic	Modify | Details
Vacuum	[redacted]	07:00:00 - 14:00:00 UTC every day	Yes	Periodic	Modify | Details

Step 5 The **Add O&M Task** page is displayed.

- Select **Vacuum** for **O&M Task**.
- Set **Scheduling Mode** to **Auto**. GaussDB(DWS) automatically scans tables that require **VACUUM** operation.
- Select **System catalogs** or **User tables** for **Autovacuum**.
  - If there are a large number of **UPDATE** and **DELETE** operations, select the **User tables**.
  - If there are a large number of **CREATE** and **DELETE** operations, select **System catalogs**.



Add O&M Task

① Specify Basic Info ② Configure Schedule ③ Finish

* O&M Task: Vacuum

Description: Enter description

Remarks: 0/256

* Scheduling Mode: Priority

Autovacuum: ● User tables (VACUUM FULL) ○ System catalogs (VACUUM)

* Vacuum First: 0/10,000

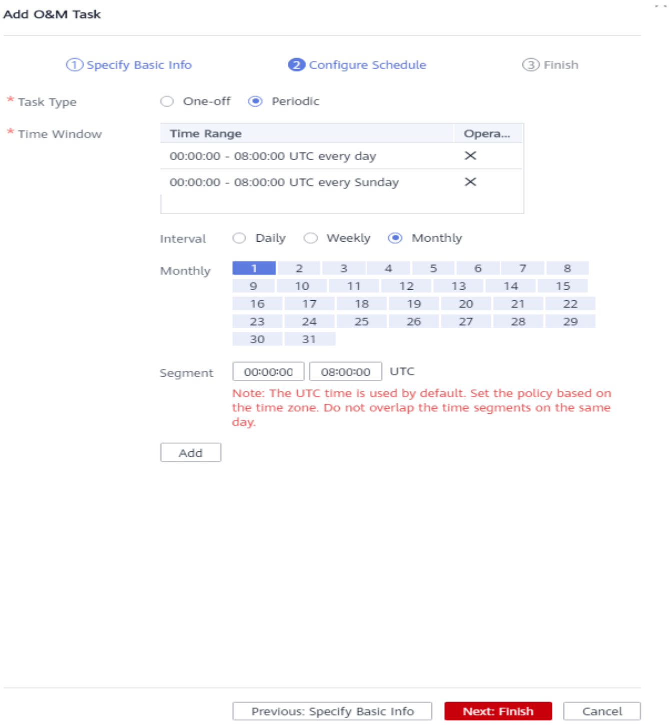
Note: Enter only one target on a single line, in the format of database1 schema1 table1. Multiple lines are allowed.

Advanced Settings: Default | Custom

Autovacuum Trigger: ☑ Table Bloat 30 %

☑ Table Reclaimable Space 100 G..

Step 6 Click **Next: Configure Schedule** to configure the schedule and Vacuum type. You are advised to select **Periodic** for **Task Type**. The GaussDB(DWS) automatically executes VACUUM in your selected time windows.



Add O&M Task

① Specify Basic Info ② Configure Schedule ③ Finish

* Task Type: ○ One-off ● Periodic

* Time Window

Time Range	Opera...
00:00:00 - 08:00:00 UTC every day	×
00:00:00 - 08:00:00 UTC every Sunday	×

Interval: ○ Daily ○ Weekly ● Monthly

Monthly

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	
16	17	18	19	20	21	22	
23	24	25	26	27	28	29	
30	31						

Segment: 00:00:00 08:00:00 UTC

Note: The UTC time is used by default. Set the policy based on the time zone. Do not overlap the time segments on the same day.

NOTE

For automatic Vacuum O&M tasks, the system uses the **VACUUM FULL** operation to process user tables. VACUUM FULL holds a level 8 lock, which blocks other transactions. Other transactions will be in lock waiting during **VACUUM FULL** execution. After 20 minutes, a timeout error is reported. Therefore, do not perform other transactions in the configured time window.

Step 7 After confirming that the information is correct, click **Next** to complete the configuration.

----End

- **8.1.2 and earlier versions: Run the VACUUM FULL command to clear data.**

NOTICE

1. **VACUUM FULL** locks a table. During **VACUUM FULL**, all accesses to the table are blocked and wait until **VACUUM FULL** ends. Please schedule properly to prevent table locking from affecting services.
2. **VACUUM FULL** is used to extract valid data from the current table and delete dirty data. This operation temporarily occupies **extra space** (the space will be released after the data is deleted). Therefore, the space increases and then decreases during **VACUUM FULL**, calculate the space required by **VACUUM FULL** in advance. (Extra space = Table size x (1 – Dirty page rate))

Step 1 Connect to the database, run the following SQL statement to query large tables whose dirty page rate exceeds 30%, and sort the tables by size in descending order:

```
SELECT schemaname AS schema, relname AS table_name, n_live_tup AS analyze_count,
pg_size_pretty(pg_table_size(relid)) as table_size, dirty_page_rate
FROM PGXC_GET_STAT_ALL_TABLES
WHERE schemaName NOT IN ('pg_toast', 'pg_catalog', 'information_schema', 'cstore', 'pmk')
AND dirty_page_rate > 30
ORDER BY table_size DESC, dirty_page_rate DESC;
```

Step 2 Check whether any command output is displayed.

- If yes, run the commands in **Step 3** for a table larger than 10 GB.
- If no, no further action is required.

Step 3 Run the **VACUUM FULL** command to clear the top 5 tables with the most dirty pages. If the maximum disk usage is greater than 70%, clear the tables one by one.

```
VACUUM FULL ANALYZE schema.table_name;
```

Step 4 If there is no table with a high dirty page rate and the disk usage is close to or exceeds 75%, expand the node or disk capacity of the cluster based on the following data warehouse types to prevent cluster read-only.

- Cloud data warehouse + SSD cloud disk: Expand the disk capacity by referring **Disk Capacity Expansion of an EVS Cluster**.

- Cloud data warehouse+local SSD disks, or old standard data warehouse (disk capacity expansion is not supported): Contact technical support for **Scaling Out a Cluster**.

----End

Scenario 2: Disk Skew and Clearing Skew Tables

If the skew rate of a table on a single DN is greater than or equal to 5%, you are advised to reselect a **distribution key** for the table and **redistribute** data.

NOTICE

1. Impacts of skewed tables: Skew tables may cause severe skew in operator computing or spilling. As a result, some DNs may be overloaded, and the advantages of GaussDB(DWS) distributed computing cannot be fully utilized, affecting system performance. Besides, they are easily to cause disk space exhaustion on a single DN.
2. In 8.1.3 and later versions, tables are created in polling (RoundRobin) mode by default (see **RoundRobin**). If you are not familiar with distribution keys, you can use the ROUNDROBIN keyword to create tables to simplify the service development (see **Application Scenarios of Polling Tables and Hash Tables**).

Step 1 Connect to the database and run the following SQL statement to query the skew tables:

```
WITH skew AS
(
        SELECT
                schemaname,
                tablename,
                pg_catalog.sum(dnsize) AS totalsize,
                pg_catalog.avg(dnsize) AS avgsize,
                pg_catalog.max(dnsize) AS maxsize,
                pg_catalog.min(dnsize) AS minsize,
                (pg_catalog.max(dnsize) - pg_catalog.min(dnsize)) AS skewsize,
                pg_catalog.stddev(dnsize) AS skewstddev
        FROM pg_catalog.pg_class c
        INNER JOIN pg_catalog.pg_namespace n ON n.oid = c.relnamespace
        INNER JOIN pg_catalog.gs_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 IN('H', 'N')
        GROUP BY schemaname,tablename
)
SELECT
        schemaname,
        tablename,
        totalsize,
        avgsize::numeric(1000),
        (maxsize/totalsize)::numeric(4,3)  AS maxratio,
        (minsize/totalsize)::numeric(4,3)  AS minratio,
        skewsize,
        (skewsize/avgsize)::numeric(4,3)  AS skewratio,
        skewstddev::numeric(1000)
FROM skew
WHERE totalsize > 0;
```

For details about how to query skew tables, see section **Data Skew Causes Slow SQL Statement Execution and Operations Fail on Large Tables**.

Step 2 Select another distribution column for the table with severe skew based on the table size and skew rate. In 8.1.0 and later versions, use the **ALTER TABLE** syntax to change the distribution column. For other versions, see **How Do I Change Distribution Columns**.

----End

Scenario 3: Cluster Read-only.

When the usage of a single disk in the cluster exceeds 90%, the system automatically triggers read-only for the cluster. In this case, if you continue to execute write services (DML or DDL), an error message similar to "ERROR: cannot execute xxx in a read-only transaction." is displayed.

Read-only is a mechanism of GaussDB(DWS) to protect user data. It prevents instance startup failures caused by 100% disk usage.

After the cluster is read-only, you need to perform operations such as **DROP** and **TRUNCATE** to delete unnecessary data as soon as possible to reduce the space usage to less than 80%. Then handle tables mentioned in scenarios 1 and 2 to prevent the disk usage increase caused by **VACUUM FULL**.

- **If the cluster version is 8.1.3 or later, perform the following steps:**

Step 1 When a cluster is in read-only status, stop the write tasks to prevent data loss caused by disk space exhaustion.

Step 2 Use the client to connect to the database, disable read-only through **START TRANSACTION**, and run the **DROP**/**TRUNCATE TABLE** command to clear unnecessary data to ensure that the disk usage is less than 80%.

Data clearing method 1:

```
START TRANSACTION READ WRITE;
drop/truncate table table_name;
COMMIT;
```

Data clearing method 2:

```
START TRANSACTION;
SET transaction_read_only=off;
drop/truncate table table_name;
COMMIT;
```

After the clearing is complete, the system automatically cancels the read-only mode.

Step 3 Check the tables mentioned in scenarios 1 and 2 to see whether there are tables that need to be handled. If no, you are advised to scale out the cluster as soon as possible. Based on the data warehouse type, scale-out is classified into node scale-out and disk scale-out.

- Cloud data warehouse + SSD cloud disk: Expand the disk capacity by referring **Disk Capacity Expansion of an EVS Cluster**.
- Cloud data warehouse+local SSD disks, or old standard data warehouse (disk capacity expansion is not supported): Contact technical support for **Scaling Out a Cluster**.

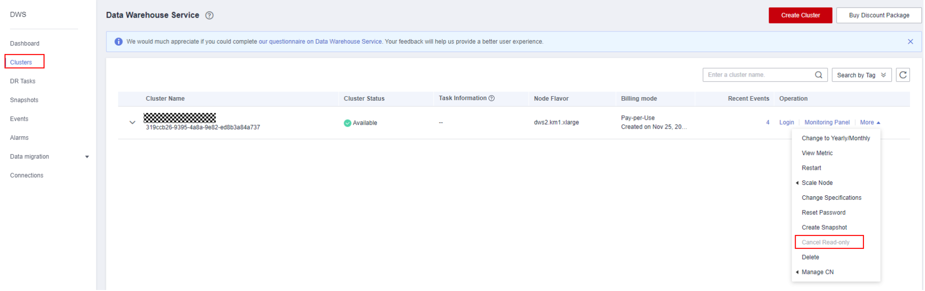
----End

- **For cluster 8.1.2 and earlier versions, perform the following steps:**

Step 1 When a cluster is in read-only status, stop the write tasks to prevent data loss caused by disk space exhaustion.

Step 2 Log in to the GaussDB(DWS) management console and cancel the read-only state of the cluster.

1. Log in to the GaussDB(DWS) management console. Click **Clusters**. All clusters will be displayed by default.
2. In the **Operation** column of the target cluster, choose **More** > **Cancel Readonly**.



Cluster Name	Cluster Status	Task Information	Node Flavor	Billing mode	Recent Events	Operation
[redacted] 319ccb26-9395-4a8a-9e82-ed8b3a84a737	Available	--	dws2.km1.xlarge	Pay-per-Use Created on Nov 25, 20...	4	Login | Monitoring Panel | More

3. In the dialog box that is displayed, click **OK** to confirm and cancel the read-only status for the cluster.

Step 3 After the read-only state is canceled, run the **DROP**/**TRUNCATE** command to delete unnecessary data. Ensure that the disk usage is less than 80%.

Step 4 Check the tables mentioned in scenarios 1 and 2 to see whether there are tables that need to be handled. If no, you are advised to scale out the cluster as soon as possible. Based on the data warehouse type, scale-out is classified into node scale-out and disk scale-out.

- Cloud data warehouse + SSD cloud disk: Expand the disk capacity by referring **Disk Capacity Expansion of an EVS Cluster**.
- Cloud data warehouse+local SSD disks, or old standard data warehouse (disk capacity expansion is not supported): Contact technical support for **Scaling Out a Cluster**.

----End

Space Management and Control

GaussDB(DWS) supports the **sql_use_spacelimit** and **temp_file_limit** parameters for statement disk space control. These parameters prevent disk usage spikes that can cause alarms or read-only mode during service operation. They also help detect services that exchange or import too much data to the database.

Step 1 Log in to the GaussDB(DWS) console, click **Clusters** on the left, and click the desired cluster. The cluster details page is displayed.

Step 2 Click **Parameters**, search for **sql_use_spacelimit** and **temp_file_limit** (see **Disk Space**) in the search box, and adjust them.

You are advised to set **sql_use_spacelimit** to 10% of the total capacity. For example, if the purchased space is 100 GB per node, set **sql_use_spacelimit** to **10 GB**.

NOTICE

After the preceding configuration takes effect, if the space usage of a SQL statement exceeds the value of this parameter, the SQL statements will be terminated. To temporarily disable the function, run the following statement in the session:

```
SET sql_use_spacelimit=0;
```

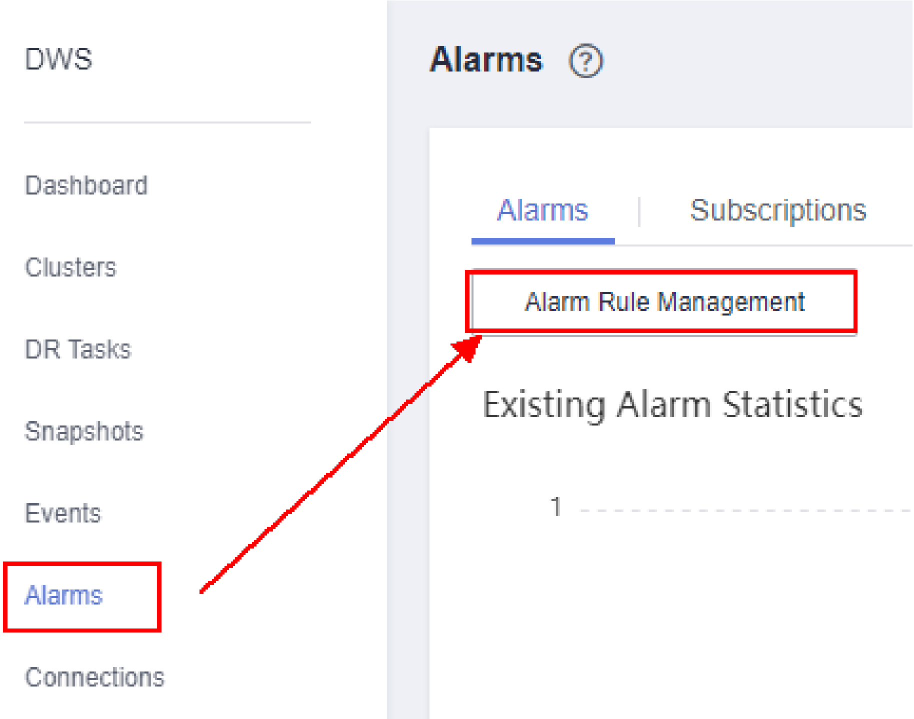
----End

Subscribing to Disk Space Alarms

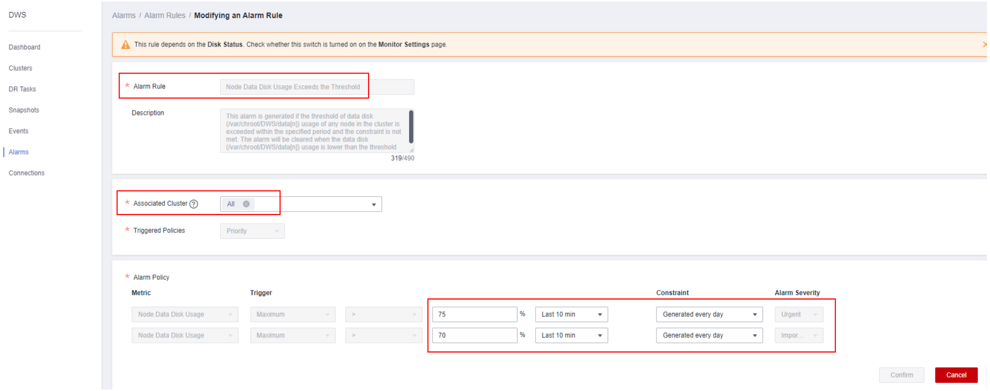
To reduce customers' O&M pressure, GaussDB(DWS) provides the alarm subscription function. When the disk usage of a cluster is greater than the preset threshold, the system notifies you through SMS messages or emails.

Step 1 Set an alarm threshold:

1. Log in to the GaussDB(DWS) console, choose **Alarms** on the left, and click **Alarm Rule Management**.



2. In the alarm rule list, click **Modify** on the right of **Node Data Disk Usage Exceeds the Threshold**.
3. In the cluster list, click the name of the target cluster. The **Cluster Information** page is displayed. Set the alarm policy as follows:
   - **Associated Cluster**: all clusters
   - **Alarm Policy**: Set the threshold of important alarms to **70%** and the duration to **Last 10 minutes**, and set the threshold of urgent alarms to **75%** and the duration to **Last 10 minutes**. Set the parameters as follows:



Alarms / Alarm Rules / **Modifying an Alarm Rule**

This rule depends on the **Disk Status**. Check whether this switch is turned on on the **Monitor Settings** page.

* Alarm Rule: Node Data Disk Usage Exceeds the Threshold

Description: This alarm is generated if the threshold of data disk (/var/chroot/DWS/data[n]) usage of any node in the cluster is exceeded within the specified period and the constraint is not met. The alarm will be cleared when the data disk (/var/chroot/DWS/data[n]) usage is lower than the threshold

319/490

* Associated Cluster: All

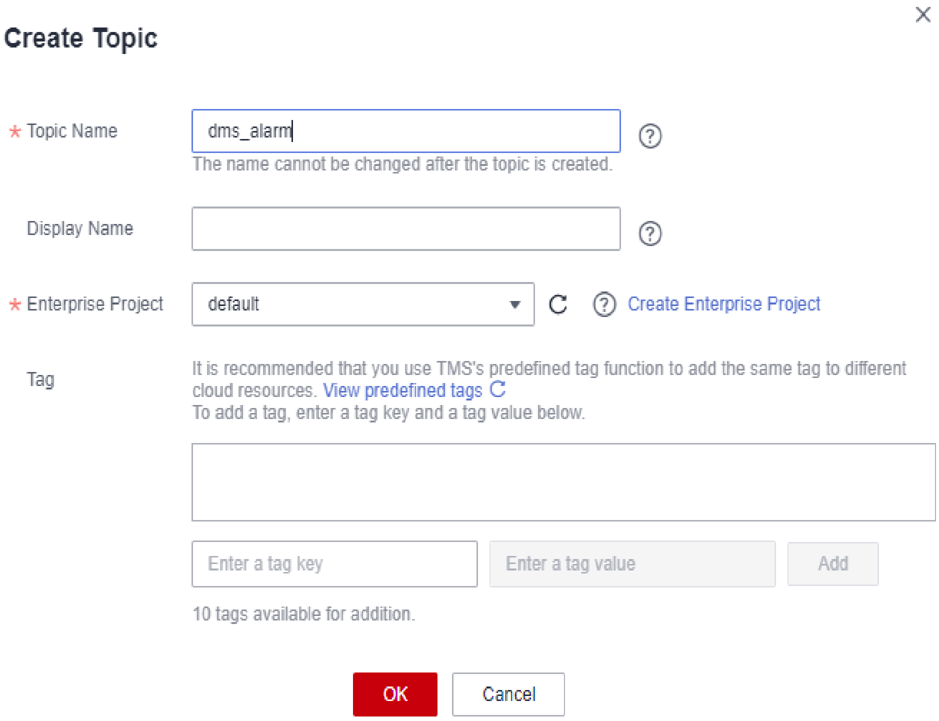
* Triggered Policies: Priority

* Alarm Policy

Metric	Trigger				Constraint	Alarm Severity
Node Data Disk Usage	Maximum	>	75 %	Last 10 min	Generated every day	Urgent
Node Data Disk Usage	Maximum	>	70 %	Last 10 min	Generated every day	Impor...

Step 2 Create a topic on the Simple Message Notification (SMN) console.

1. Switch to the SMN console and click **Create Topic**. Set **Topic Name** and **Enterprise Project** as follows.



Create Topic

* Topic Name: dms_alarm

The name cannot be changed after the topic is created.

Display Name:

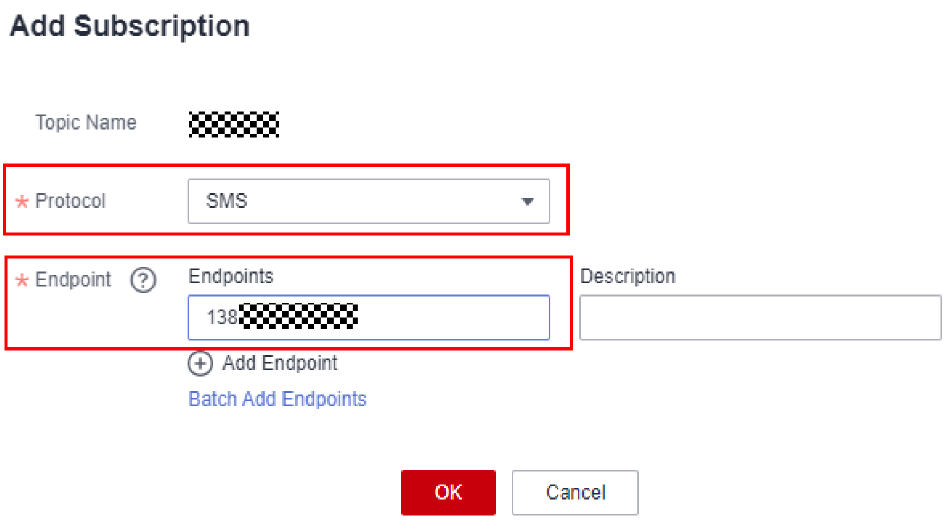
* Enterprise Project: default

Tag: It is recommended that you use TMS's predefined tag function to add the same tag to different cloud resources. View predefined tags

To add a tag, enter a tag key and a tag value below.

10 tags available for addition.

2. After the topic is created, click **Add Subscription** on the right. Select SMS or Email, and enter the mobile number or email address in the **Endpoint** text box.



Add Subscription

Topic Name [redacted]

* Protocol SMS

* Endpoint

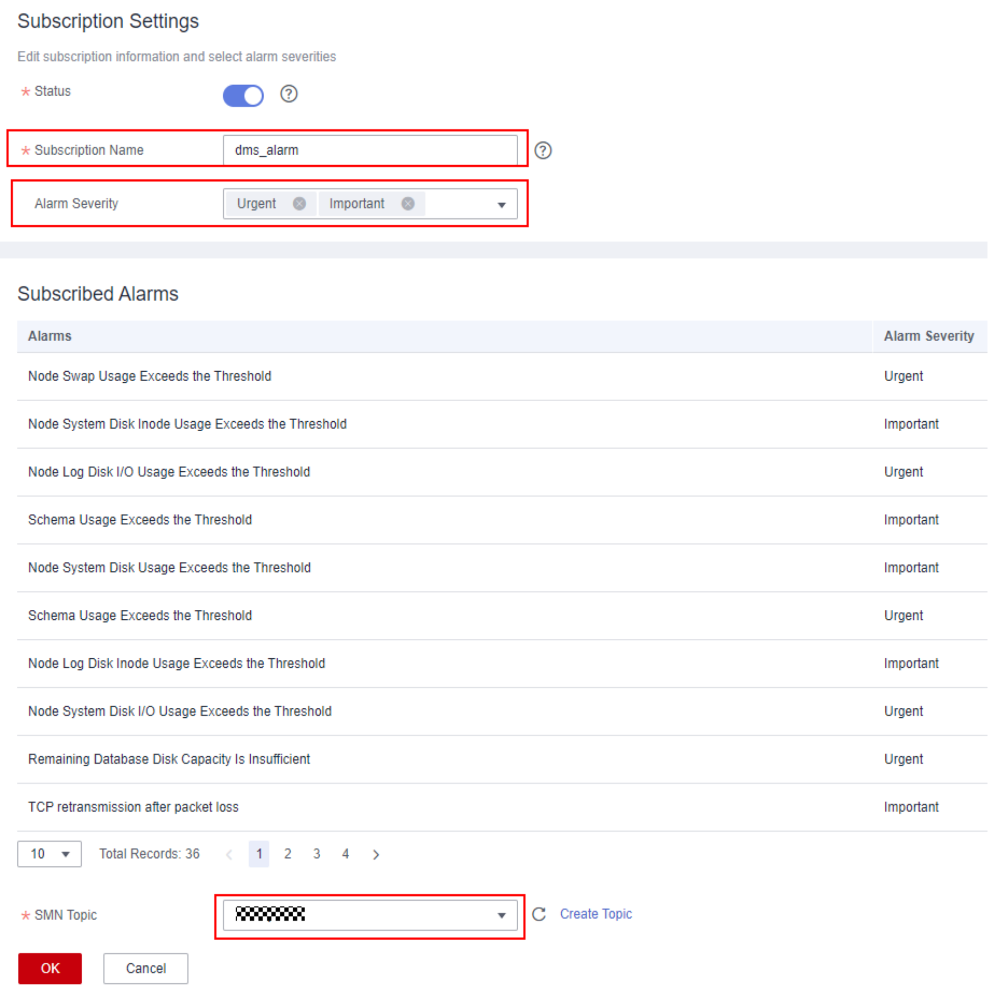
Endpoints: 138[redacted]

Description:

3. The entered mobile number or email address will receive a confirmation SMS message or email. Click **OK** to complete the subscription.

Step 3 Add an alarm subscription.

1. Return to the GaussDB(DWS) console, choose **Alarms** on the left, click **Subscriptions**, and click **Create Subscription**.
2. Set **Subscription Name** to **dms_alarm**, **Alarm Severity** to **Urgent** and **Important**, and **SMN Topic** to **dms_alarm** created in the previous step.



Subscription Settings

Edit subscription information and select alarm severities

* Status

* Subscription Name: dms_alarm

Alarm Severity: Urgent, Important

Subscribed Alarms

Alarms	Alarm Severity
Node Swap Usage Exceeds the Threshold	Urgent
Node System Disk Inode Usage Exceeds the Threshold	Important
Node Log Disk I/O Usage Exceeds the Threshold	Urgent
Schema Usage Exceeds the Threshold	Important
Node System Disk Usage Exceeds the Threshold	Important
Schema Usage Exceeds the Threshold	Urgent
Node Log Disk Inode Usage Exceeds the Threshold	Important
Node System Disk I/O Usage Exceeds the Threshold	Urgent
Remaining Database Disk Capacity Is Insufficient	Urgent
TCP retransmission after packet loss	Important

* SMN Topic: [redacted]

3. Click **OK**. After the alarm subscription is configured, a notification will be sent when the disk usage exceeds 70% or 75%.

----End

6.5 SQL Execution Is Slow with Low Performance and Sometimes Does Not End After a Long Period of Time

Symptom

The SQL execution is slow with low performance and sometimes does not end after a long period of time.

Possible Causes

Analyze the causes of slow SQL execution from the following aspects:

1. Run the **EXPLAIN** command to view the SQL execution plan and determine whether to optimize the SQL statements based on the plan.
2. Check whether the query is blocked. If the query is blocked, the statement execution takes a long time. In this case, you can forcibly terminate the abnormal sessions.

3. Review and modify the table definitions. Select an appropriate distribution key to avoid data skew.
4. Check whether the SQL statements use functions that do not support pushdown. You are advised to use the syntax or function that supports pushdown.
5. Run the **VACUUM FULL** and **ANALYZE** commands periodically to reclaim the disk space occupied by updated or deleted data.
6. Check whether the table has an index. You are advised to re-create the index regularly.

   After multiple deletion operations are performed on the database, the index key on the index page will be deleted, causing index expansion. Re-creating the index regularly can improve the query efficiency.
7. Optimize services and analyze whether large tables can be divided.

Handling Procedure

GaussDB(DWS) provides methods for analyzing and optimizing queries, as well as some common cases and error handling methods. Common problem locating methods are as follows:

- Method 1: Periodically collect statistics on the table and optimize the table data.

  If you frequently run the INSERT statement to insert data into a table, you need to periodically run the ANALYZE statement on the table.

  ```
  ANALYZE table_name;
  ```

  If you frequently run the DELETE statement to delete data from a table, you need to periodically run the VACUUM FULL statement on the table.

  ```
  Vacuum full table_name;
  ```

  NOTE

  VACUUM FULL cannot be used when other tasks are running.

  Query the table size. If the table size is large but only a small amount of data exists, run the **VACUUM FULL** command to reclaim the space.

  ```
  SELECT * FROM pg_size_pretty(pg_table_size('tablename'));
  Vacuum full table_name;
  ```

Method 2: Query information about running SQL statements in the **PGXC_STAT_ACTIVITY** view.

Step 1 Run the following command to view the information about the SQL statements that are not in the idle state:

```
SELECT pid,datname,usename,state,waiting,query FROM pgxc_stat_activity WHERE state <> 'idle';
```

Step 2 Run the following command to view blocked query statements:

```
SELECT pid,datname, usename, state,waiting,query FROM pgxc_stat_activity WHERE state <> 'idle' and
waiting=true;
```

Step 3 Check whether the query statement is blocked.

- If no blocking occurs, search for related service tables and rectify the fault according to **Method 1**.
- If a statement is blocked, end the blocked statement based on the thread ID of the faulty session.

```
SELECT pg_terminate_backend(pid);
```

----End

6.6 Data Skew Causes Slow SQL Statement Execution and Operations Fail on Large Tables

Symptom

SQL statement execution is slow and SQL statements cannot be executed on large tables.

Possible Causes

The distribution modes supported by GaussDB(DWS) are hash, replication, and roundrobin (supported by 8.1.2 clusters and later versions). If the created table is distributed in Hash mode and the distribution key is not specified, the first column of the table is selected as the distribution key. In this case, skew may occur. Table skew has the following negative impacts.

- The SQL execution performance is poor because data is distributed only on some DNs. When the SQL statement is executed, only some DNs are involved in computing, and the advantage of distributed computing is not leveraged.
- The usage of resources, especially disks, will be skewed. That is, the usage of some disks may be close to the upper limit, but the usage of other disks is low.
- The CPU usage of some nodes may be excessively high.

Cause Analysis

Step 1 Log in to the GaussDB(DWS) management console. On the **Clusters** page, locate the target cluster. In the **Operation** column of the target cluster, click **Monitoring Panel**. Choose **Monitoring** > **Node Monitoring**. Click the **Disks** tab to view the disk usage.

NOTE

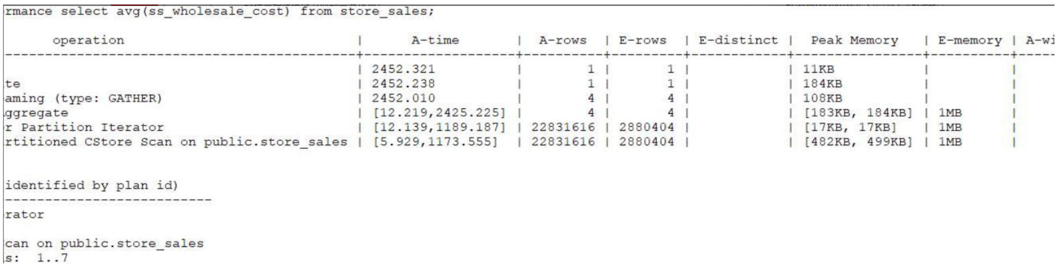
Check the usage of each data disk. It is found that the usage is uneven among data disks. Generally, the difference between the highest and the lowest disk usage is small. If the difference exceeds 5%, data skew may occur.

Step 2 Connect to the database and check the job operating status in the waiting view. It is found that the job waits for being processed by one or some DNs.

```
SELECT wait_status, count(*) as cnt FROM pgxc_thread_wait_status WHERE wait_status not like '%cmd%'
AND wait_status not like '%none%' and wait_status not like '%quit%' group by 1 order by 2 desc;
```

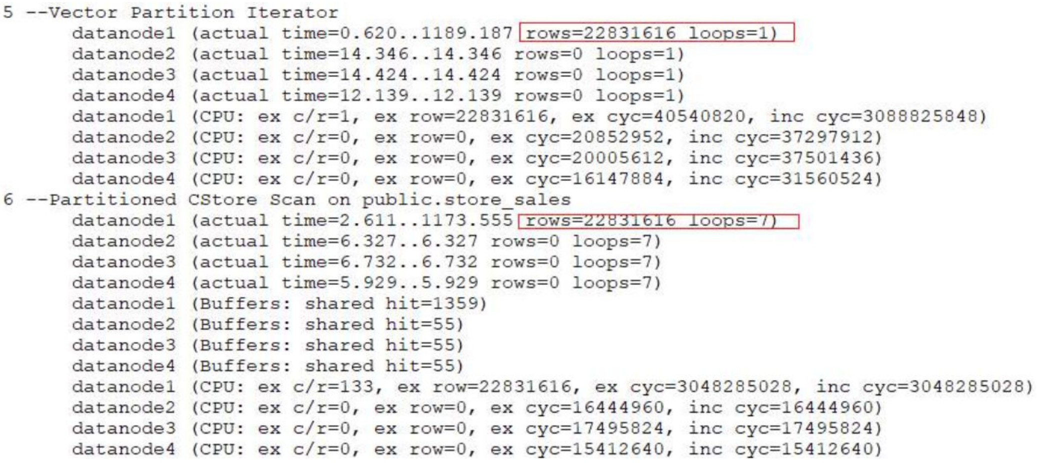
Step 3 The **explain performance** of the slow statement shows that the scan time and number of scan rows in the base table of each DN are unbalanced.

```
explain performance select avg(ss_wholesale_cost) from store_sales;
```



```
rmance select avg(ss_wholesale_cost) from store_sales;

      operation                                |      A-time        |  A-rows  | E-rows  | E-distinct |  Peak Memory   | E-memory | A-wi
-----------------------------------------------+--------------------+----------+---------+------------+----------------+----------+-----
                                               | 2452.321           |        1 |       1 |            | 11KB           |          |
te                                             | 2452.238           |        1 |       1 |            | 184KB          |          |
aming (type: GATHER)                           | 2452.010           |        4 |       4 |            | 108KB          |          |
ggregate                                       | [12.219,2425.225]  |        4 |       4 |            | [183KB, 184KB] | 1MB      |
r Partition Iterator                           | [12.139,1189.187]  | 22831616 | 2880404 |            | [17KB, 17KB]   | 1MB      |
rtitioned CStore Scan on public.store_sales    | [5.929,1173.555]   | 22831616 | 2880404 |            | [482KB, 499KB] | 1MB      |


identified by plan id)
--------------------------
rator

can on public.store_sales
s:  1..7
```

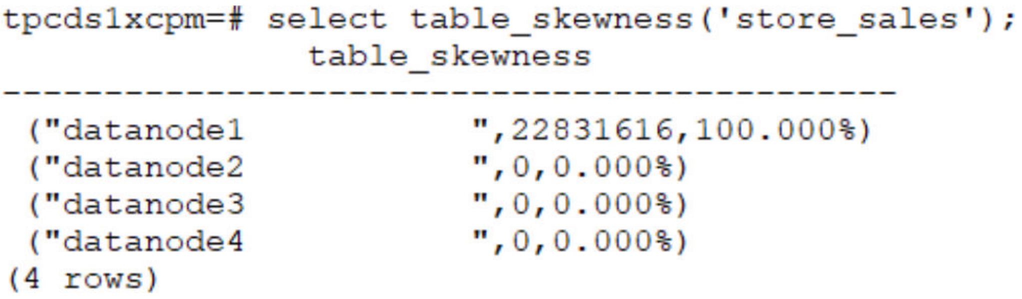
- Time of scanning a base table: The fastest DN takes 5 ms, and the slowest DN takes 1173 ms.
- Data distribution: Some DNs have 22,831,616 rows and other DNs have no row, resulting in data skew.



```
5 --Vector Partition Iterator
      datanode1 (actual time=0.620..1189.187 rows=22831616 loops=1)
      datanode2 (actual time=14.346..14.346 rows=0 loops=1)
      datanode3 (actual time=14.424..14.424 rows=0 loops=1)
      datanode4 (actual time=12.139..12.139 rows=0 loops=1)
      datanode1 (CPU: ex c/r=1, ex row=22831616, ex cyc=40540820, inc cyc=3088825848)
      datanode2 (CPU: ex c/r=0, ex row=0, ex cyc=20852952, inc cyc=37297912)
      datanode3 (CPU: ex c/r=0, ex row=0, ex cyc=20005612, inc cyc=37501436)
      datanode4 (CPU: ex c/r=0, ex row=0, ex cyc=16147884, inc cyc=31560524)
6 --Partitioned CStore Scan on public.store_sales
      datanode1 (actual time=2.611..1173.555 rows=22831616 loops=7)
      datanode2 (actual time=6.327..6.327 rows=0 loops=7)
      datanode3 (actual time=6.732..6.732 rows=0 loops=7)
      datanode4 (actual time=5.929..5.929 rows=0 loops=7)
      datanode1 (Buffers: shared hit=1359)
      datanode2 (Buffers: shared hit=55)
      datanode3 (Buffers: shared hit=55)
      datanode4 (Buffers: shared hit=55)
      datanode1 (CPU: ex c/r=133, ex row=22831616, ex cyc=3048285028, inc cyc=3048285028)
      datanode2 (CPU: ex c/r=0, ex row=0, ex cyc=16444960, inc cyc=16444960)
      datanode3 (CPU: ex c/r=0, ex row=0, ex cyc=17495824, inc cyc=17495824)
      datanode4 (CPU: ex c/r=0, ex row=0, ex cyc=15412640, inc cyc=15412640)
```

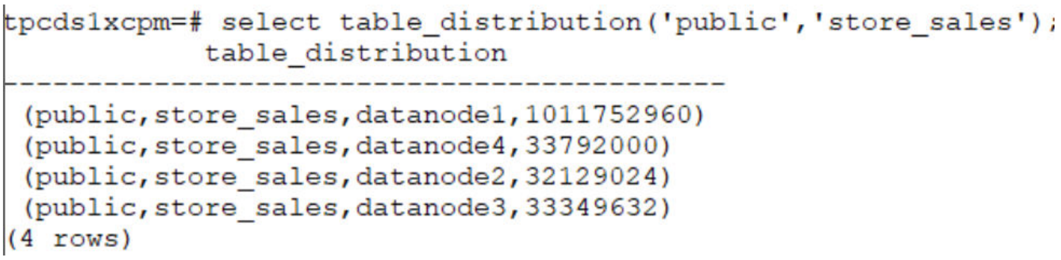
Step 4 You can detect data skew by using the skew check interface.

```
SELECT table_skewness('store_sales');
```



```
tpcds1xcpm=# select table_skewness('store_sales');
               table_skewness
--------------------------------------------
 ("datanode1          ",22831616,100.000%)
 ("datanode2          ",0,0.000%)
 ("datanode3          ",0,0.000%)
 ("datanode4          ",0,0.000%)
(4 rows)
```

```
SELECT table_distribution('public','store_sales');
```



```
tpcds1xcpm=# select table_distribution('public','store_sales');
              table_distribution
---------------------------------------
 (public,store_sales,datanode1,1011752960)
 (public,store_sales,datanode4,33792000)
 (public,store_sales,datanode2,32129024)
 (public,store_sales,datanode3,33349632)
(4 rows)
```

Step 5 The resource monitoring result shows that the CPU usage and I/O of some nodes are significantly higher than those of other nodes.

----End

Handling Procedure

How to find the skewed table

1. If the number of tables in the database is less than 10,000, use the **PGXC_GET_TABLE_SKEWNESS** view to query data skew of all tables in the database.

   ```
   SELECT * FROM pgxc_get_table_skewness ORDER BY totalsize DESC;
   ```

2. If the number of tables in the database is greater than 10,000, it may take a long time (hours) to query the entire database and calculate skew columns in the view. You are advised to perform the following operations by referring to the definition of the **PGXC_GET_TABLE_SKEWNESS** view:

   - In 8.1.2 and earlier cluster versions, the function is used to optimize calculation by customizing output and reducing output columns. For example:

     ```
     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;
     ```

   - For clusters of 8.1.3 and later versions, the function can be used to check data skew of all tables in the database. The **gs_table_distribution()** function is better than the **table_distribution()** function when you query all tables in the database. In a large cluster with a large amount of data, use the **gs_table_distribution()** function.

     ```
     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.gs_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;
     ```

NOTE

Run the following statement to query large tables:

```
SELECT schemaname||'.'||tablename as table, sum(dnsize) as size FROM
gs_table_distribution() group by 1 order by 2 desc limit 10;
```

Run the following statement to query the table skew rate:

```
WITH skew AS
(
        SELECT
                schemaname,
                tablename,
                pg_catalog.sum(dnsize) AS totalsize,
                pg_catalog.avg(dnsize) AS avgsize,
                pg_catalog.max(dnsize) AS maxsize,
                pg_catalog.min(dnsize) AS minsize,
                (pg_catalog.max(dnsize) - pg_catalog.min(dnsize)) AS skewsize,
                pg_catalog.stddev(dnsize) AS skewstddev
        FROM pg_catalog.pg_class c
        INNER JOIN pg_catalog.pg_namespace n ON n.oid = c.relnamespace
        INNER JOIN pg_catalog.gs_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 IN('H',
'N')
        GROUP BY schemaname,tablename
)
SELECT
        schemaname,
        tablename,
        totalsize,
        avgsize::numeric(1000),
        (maxsize/totalsize)::numeric(4,3)  AS maxratio,
        (minsize/totalsize)::numeric(4,3)  AS minratio,
        skewsize,
        (skewsize/avgsize)::numeric(4,3)  AS skewratio,
        skewstddev::numeric(1000)
FROM skew
WHERE totalsize > 0;
```

Methods of selecting a distribution key for a table

1. If the **distinct** value of the column is large and no obvious data skew occurs, you can define multiple columns as a distribution key.

   View the size of **distinct**.

   ```
   SELECT count(distinct column1) FROM table;
   ```

   Check whether data skew occurs.

   ```
   SELECT count(*) cnt, column1 FROM table group by column1 order by cnt limit 100;
   ```

2. Select the columns where **JOIN** or **GROUP BY** statement is frequently used to reduce the use of **STREAM**.
3. Some unrecommended operations are as follows:
   a. The default value of the distribution key (the first column) is used.
   b. The distribution key is generated through the auto-increment of sequences.
   c. The distribution key is generated using a random number. This method is recommended only when any column or any combination of two columns is skewed.

6.7 Table Size Does not Change After VACUUM FULL Is Executed on the Table

Symptom

A user runs the **VACUUM FULL** command to clear a table, but the table size does not change.

Possible Causes

Assume the table is name table_name. Possible causes are as follows:

1. No data has been deleted from the table. Therefore, **VACUUM FULL table_name** has nothing to delete, causing that the table size does not change.
2. Concurrent transactions exist during the execution of **VACUUM FULL table_name**, causing the recently deleted data to be skipped. As a result, the table size does not change.

Solution

The following are solutions for the second possible cause:

- Wait until all concurrent transactions are complete, and then run the **VACUUM FULL table_name** command again.
- If the table file size remains unchanged after the preceding operations are performed, ensure that no task is running in the cluster and all data has been saved. Then, perform the following operations:

Step 1 Run the following command to query the current transaction XID.

```
SELECT txid_current();
```

Step 2 Run the following command to view the active transaction list.

```
SELECT txid_current_snapshot();
```

Step 3 If the XID of any transaction in the active transaction list is smaller than that of the current transaction, restart the cluster and run the **VACUUM FULL** statement to clear the table again.

----End

6.8 VACUUM Is Executed After Table Data Deletion, But the Space Is Not Released

Symptom

After a user deletes the data in a table and executes **VACUUM**, the storage space is not released.

Possible Causes

- The user may not have the permission for executing **VACUUM** on some tables or the database does not have too much data expansion.
- By default, **VACUUM** clears only the tables on which the current user has permissions in the database.
- The **vacuum_defer_cleanup_age** parameter is not set to **0**. In earlier versions, the default value of this parameter is 8000, indicating that dirty data generated by the latest 8000 transactions is not cleared.
- Dirty data generated by the transactions whose ID is greater than that of the currently active transactions is not cleared to ensure transaction visibility.

Handling Procedure

- Run **VACUUM FULL** on a single table. The command format is **VACUUM FULL** *Table_name*.
- If you do not have the permission on the table, contact the database administrator or the table owner.
- If the value of **vacuum_defer_cleanup_age** is not **0**, set this parameter to **0** to cancel the transaction delay of **VACUUM**.
- If old transactions exist, restart the cluster and run the **VACUUM FULL** command again, which can ensure that space is reclaimed. Otherwise, run the **VACUUM FULL** command only after the old transactions are complete.

6.9 Error LOCK_WAIT_TIMEOUT Is Reported When VACUUM FULL Is Executed

Symptom

The following error is reported when the **VACUUM FULL** command is executed:

```
[0]ERROR: dn_6009_6010: Lock wait timeout: thread 140158632457984 on node dn_6009_6010 waiting
for AccessExclusiveLock on relation 2299036 of database 14522 after 1202001.968 ms
Detail: blocked by hold lock thread 140150147380992, statement <<backend information not available>>,
hold lockmode AccessShareLock.
Line Number: 1
```

Possible Causes

"Lock wait timeout" in the log indicates that the lockwait times out. Lock wait timeout is generally caused by the fact that another SQL statement has held the lock. The current SQL statement can be executed only after the SQL statement that holds the lock is successfully executed and releases the lock. If the lock wait time exceeds the specified value of the GUC parameter **lockwait_timeout**, the system reports the LOCK_WAIT_TIMEOUT error.

VACUUM FULL command execution may cause this error. For example, if you run the command over the entire database, the execution time may be long and may time out.

Handling Procedure

Run the **VACUUM FULL** command over a single table. The command format is *VACUUM FULL table name*. In addition, increase the frequency for running the command. Especially for tables that are frequently added, deleted, or modified, run the **VACUUM FULL** command periodically.

6.10 VACUUM FULL Is Slow

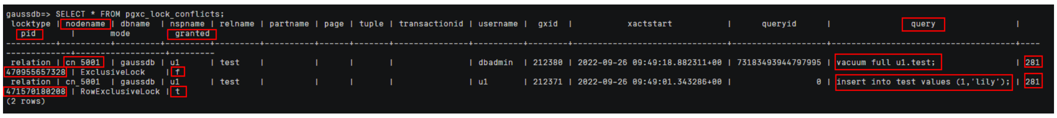
Common scenarios and troubleshooting methods for slow execution of **VACUUM FULL** are as follows:

Scenario 1: VACUUM FULL Is Executed Slowly Due to Lock Wait

- **If the cluster version is 8.1.x or later, perform the following steps:**

Step 1 Query the **pgxc_lock_conflicts** view to check lock conflicts.

```
SELECT * FROM pgxc_lock_conflicts;
```



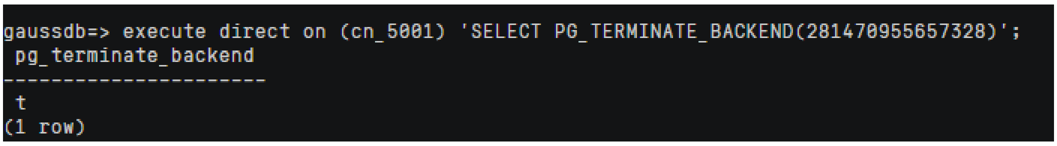
```
gaussdb=> SELECT * FROM pgxc_lock_conflicts;
 locktype | nodename | dbname  | nspname | relname | partname | page | tuple | transactionid | username |  gxid  |          xactstart            |      queryid      |               query                 |
  pid         |       mode        granted
----------+----------+---------+---------+---------+----------+------+-------+---------------+----------+--------+-------------------------------+-------------------+-------------------------------------+----
-------------+-------------------+---------
 relation | cn_5001  | gaussdb | u1      | test    |          |      |       |               | dbadmin  | 212380 | 2022-09-26 09:49:18.882311+00 | 73183493944797995 | vacuum full u1.test;                | 281
470955657328 | ExclusiveLock     | f
 relation | cn_5001  | gaussdb | u1      | test    |          |      |       |               | u1       | 212371 | 2022-09-26 09:49:01.343286+00 |                 0 | insert into test values (1,'lily'); | 281
471570180208 | RowExclusiveLock  | t
(2 rows)
```

- As shown in the following figure, if the value of **granted** is **f**, the **VACUUM FULL** statement is waiting for another lock. If **granted** is **t**, the **INSERT** statement holds the lock. **nodename** indicates the CN or DN where the lock is generated, for example, cn_5001. Execute **2**.
- If 0 rows is displayed in the command output, no lock conflict occurs. In this case, check other scenarios.

Step 2 Based on the statement content, determine whether to run **VACUUM FULL** immediately after terminating the lock-holding statement or run **VACUUM FULL** during off-peak hours.

To terminate a lock-holding statement, run the following statement: In the preceding command, *pid* is obtained from step 1, and **cn_5001** is the node name queried out.

```
execute direct on (cn_5001) 'SELECT PG_TERMINATE_BACKEND(pid)';
```



```
gaussdb=> execute direct on (cn_5001) 'SELECT PG_TERMINATE_BACKEND(281470955657328)';
 pg_terminate_backend
----------------------
 t
(1 row)
```

Step 3 After the statement is terminated, run **VACUUM FULL** again.

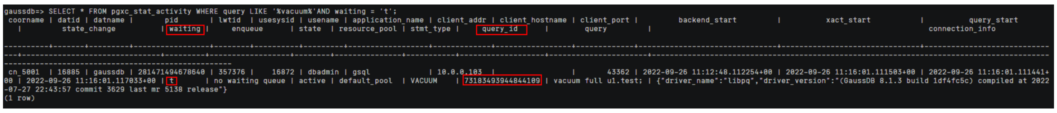
```
VACUUM FULL table_name;
```

----End

- **For 8.0.x and earlier versions:**

Step 1 Run the following statement in the database to obtain the value of **query_id** corresponding to the **VACUUM FULL** operation:

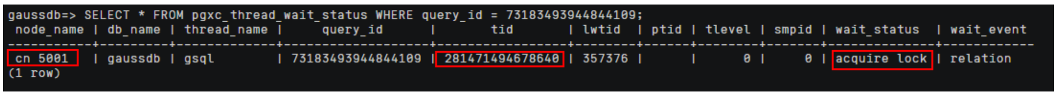
```
SELECT * FROM pgxc_stat_activity WHERE query LIKE '%vacuum%'AND waiting = 't';
```



```
gaussdb=> SELECT * FROM pgxc_stat_activity WHERE query LIKE '%vacuum%'AND waiting = 't';
 coorname | datid | datname |       pid       | lwtid  | usesysid | usename | application_name | client_addr | client_hostname | client_port |          backend_start          |           xact_start           |          query_start
   |          state_change          | waiting |     enqueue      | state  | resource_pool | stmt_type |      query_id      |         query         |                                              connection_info

----------+-------+---------+-----------------+--------+----------+---------+------------------+-------------+-----------------+-------------+-------------------------------+-------------------------------+------------------------------
---+-------------------------------+---------+------------------+--------+---------------+-----------+-------------------+----------------------+------------------------------------------------------------------
---------------------------------------------------
 cn_5001  | 16885 | gaussdb | 281471494678640 | 357376 |    16872 | dbadmin | gsql             | 10.0.0.103  |                 |       43362 | 2022-09-26 11:12:48.112254+00 | 2022-09-26 11:16:01.111503+00 | 2022-09-26 11:16:01.111441+
00 | 2022-09-26 11:16:01.117033+00 | t       | no waiting queue | active | default_pool  | VACUUM    | 73183493944844109 | vacuum full u1.test; | {"driver_name":"libpq","driver_version":"(GaussDB 8.1.3 build 1df4fc5c) compiled at 2022
-07-27 22:43:57 commit 3629 last mr 5138 release"}
(1 row)
```

Step 2 Run the following statement to check whether lock wait exists (Use the obtained **query_id**):

```
SELECT * FROM pgxc_thread_wait_status WHERE query_id = {query_id};
```



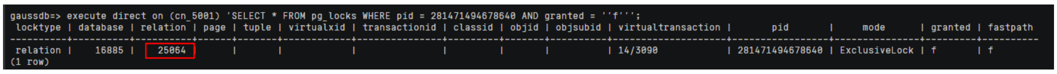
```
gaussdb=> SELECT * FROM pgxc_thread_wait_status WHERE query_id = 73183493944844109;
 node_name | db_name | thread_name |     query_id      |       tid       | lwtid  | ptid | tlevel | smpid | wait_status  | wait_event
-----------+---------+-------------+-------------------+-----------------+--------+------+--------+-------+--------------+------------
 cn_5001   | gaussdb | gsql        | 73183493944844109 | 281471494678640 | 357376 |      |      0 |     0 | acquire lock | relation
(1 row)
```

- If **acquire lock** is displayed in **wait_status** in the command output, lock wait exists. Check the value of node_name and record it, for example, cn_5001 or dn_600x_600y, then go to **Step 3**.
- If in the query result, **wait_status** does not contain **acquire lock**, there is no lock wait. Check other scenarios.

Step 3 Run the following statement to check the lock waited by the **VACUUM FULL** operation in **pg_locks:** The following uses **cn_5001** as an example. If the lock wait is on a DN, change it to the corresponding DN name. **pid** is obtained in **Step 2**.

Record the value of **relation** in the command output.

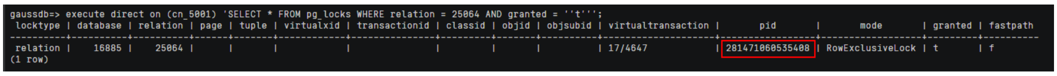
```
execute direct on (cn_5001) 'SELECT * FROM pg_locks WHERE pid = {tid} AND granted = ''f''';
```



```
gaussdb=> execute direct on (cn_5001) 'SELECT * FROM pg_locks WHERE pid = 281471494678640 AND granted = ''f''';
 locktype | database | relation | page | tuple | virtualxid | transactionid | classid | objid | objsubid | virtualtransaction |       pid       |     mode      | granted | fastpath
----------+----------+----------+------+-------+------------+---------------+---------+-------+----------+--------------------+-----------------+---------------+---------+----------
 relation |    16885 |    25064 |      |       |            |               |         |       |          | 14/3090            | 281471494678640 | ExclusiveLock | f       | f
(1 row)
```

Step 4 View in **pg_locks** the PID of the transaction holding the lock based on the value of **relation** . The value of **relation** is obtained in **Step 3**.

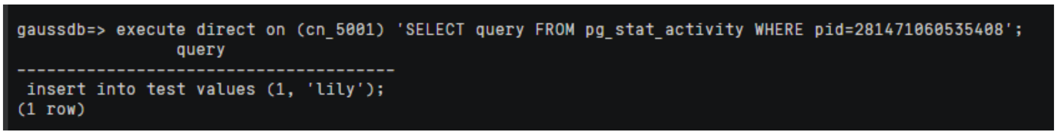
```
execute direct on (cn_5001) 'SELECT * FROM pg_locks WHERE relation = {relation} AND granted = ''t''';
```



```
gaussdb=> execute direct on (cn_5001) 'SELECT * FROM pg_locks WHERE relation = 25064 AND granted = ''t''';
 locktype | database | relation | page | tuple | virtualxid | transactionid | classid | objid | objsubid | virtualtransaction |       pid       |       mode       | granted | fastpath
----------+----------+----------+------+-------+------------+---------------+---------+-------+----------+--------------------+-----------------+------------------+---------+----------
 relation |    16885 |    25064 |      |       |            |               |         |       |          | 17/4647            | 281471060535408 | RowExclusiveLock | t       | f
(1 row)
```

Step 5 Run the following statement to query the corresponding SQL statement based on the **PID**: The value of **pid** is obtained in **Step 4**.

```
execute direct on (cn_5001) 'SELECT query FROM pg_stat_activity WHERE pid ={pid}';
```

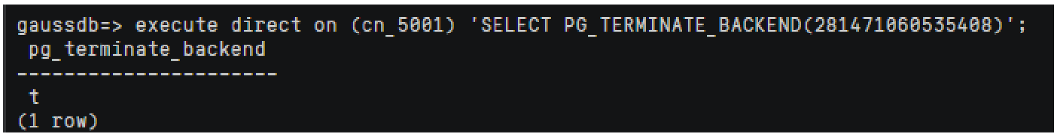


```
gaussdb=> execute direct on (cn_5001) 'SELECT query FROM pg_stat_activity WHERE pid=281471060535408';
                 query
--------------------------------------
 insert into test values (1, 'lily');
(1 row)
```

Step 6 Based on the statement content, determine whether to run **VACUUM FULL** immediately after terminating the lock-holding statement or run **VACUUM FULL** during off-peak hours.

To terminate a lock-holding statement, run the following statement: In the preceding command, *pid* is obtained from step 1, and **cn_5001** is the node name queried in **Step 4**.

```
execute direct on (cn_5001) 'SELECT PG_TERMINATE_BACKEND(pid)';
```



```
gaussdb=> execute direct on (cn_5001) 'SELECT PG_TERMINATE_BACKEND(281471060535408)';
 pg_terminate_backend
----------------------
 t
(1 row)
```

Step 7 After the statement is terminated, run the **VACUUM FULL** statement.

```
VACUUM FULL table_name;
```

----End

Scenario 2: Transactions Cannot Be Committed Due to I/O or Network Problems

Solution: Run a simple **CRETAE TABLE** statement. If the execution of the **CRETAE TABLE** statement is also slow, the I/O or network may be faulty. In this case, check the I/O and network conditions.

Scenario 3: VACUUM FULL Is Executed Slowly Due to Large System Catalogs

After fixing the I/O or network problems, run **VACUUM FULL** on empty tables. If **VACUUM FULL** is executed slowly even on empty tables, the system catalogs are too large. Executing **VACUUM FULL** on any table requires scanning the system catalogs **pg_class**, **pg_partition**, and **pg_proc**. If the three system catalogs are too large, **VACUUM FULL** will be executed slowly.

Solution: GaussDB(DWS) supports executing **VACUUM FULL** on system catalogs. However, the execution holds an eight-level lock, and services related to system catalogs will be blocked. Clear system catalogs during off-peak hours or when services are stopped and no DDL operation is performed.

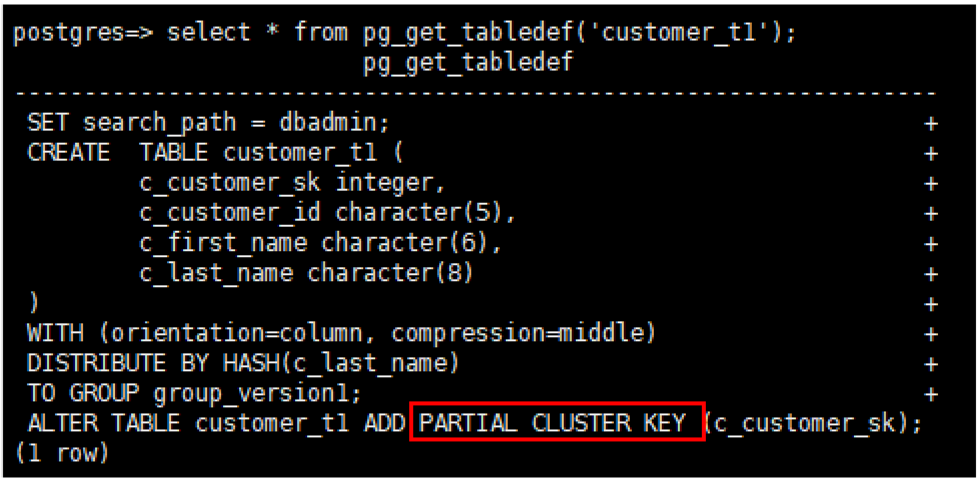
Scenario 4: Slow VACUUM FULL on a Column-store Table Using Partial Clustering (PCK)

When **VACUUM FULL** is performed on a column-store table, if PCK exists, all records in **PARTIAL_CLUSTER_ROWS** are loaded to the memory and then sorted. If the table is large or **psort_work_mem** is set to a small value, data will spill to disks during PCK sorting (the database stores temporary results to disks). This increases the time consumption.

Solution: Adjust the values of **PARTIAL_CLUSTER_ROWS** and **psort_work_mem** based on the tuple length of the data in the table.

1. Run the following statement to view the table definition. If **PARTIAL CLUSTER KEY** is displayed in the command output, the table contains PCKs.

```
SELECT * FROM pg_get_tabledef('table name');
```



```
postgres=> select * from pg_get_tabledef('customer_t1');
                         pg_get_tabledef
-------------------------------------------------------------------
 SET search_path = dbadmin;                                       +
 CREATE  TABLE customer_t1 (                                      +
         c_customer_sk integer,                                   +
         c_customer_id character(5),                              +
         c_first_name character(6),                               +
         c_last_name character(8)                                 +
 )                                                                +
 WITH (orientation=column, compression=middle)                    +
 DISTRIBUTE BY HASH(c_last_name)                                  +
 TO GROUP group_version1;                                         +
 ALTER TABLE customer_t1 ADD PARTIAL CLUSTER KEY (c_customer_sk);
(1 row)
```

2. Log in to the GaussDB(DWS) management console and choose **Clusters** on the left.
3. Click the cluster name to go to the cluster details page.
4. Click **Parameters** on the left, enter **psort_work_mem** in the search box, increase the values on both CNs and DNs, and click **Save**.
5. Run **VACUUM FULL** again.

6.11 Table Bloating Causes Slow SQL Query and Failed Data Loading on the GUI

Symptom

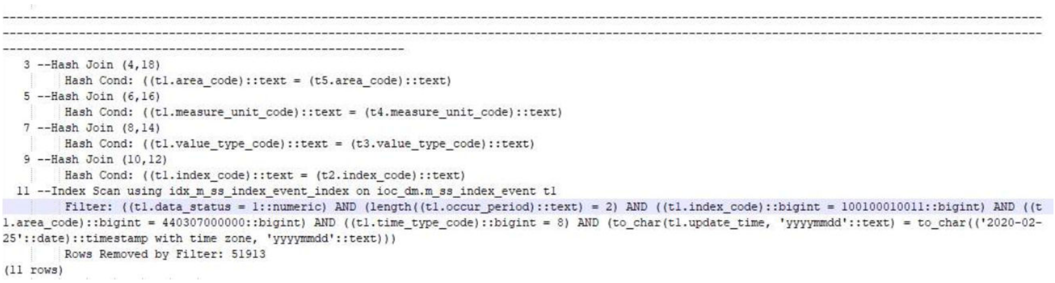
The results of SQL statements that can be executed within seconds are not generated after more than 20 seconds. As a result, data loading on the GUI times out and charts cannot be displayed for users.

Possible Causes

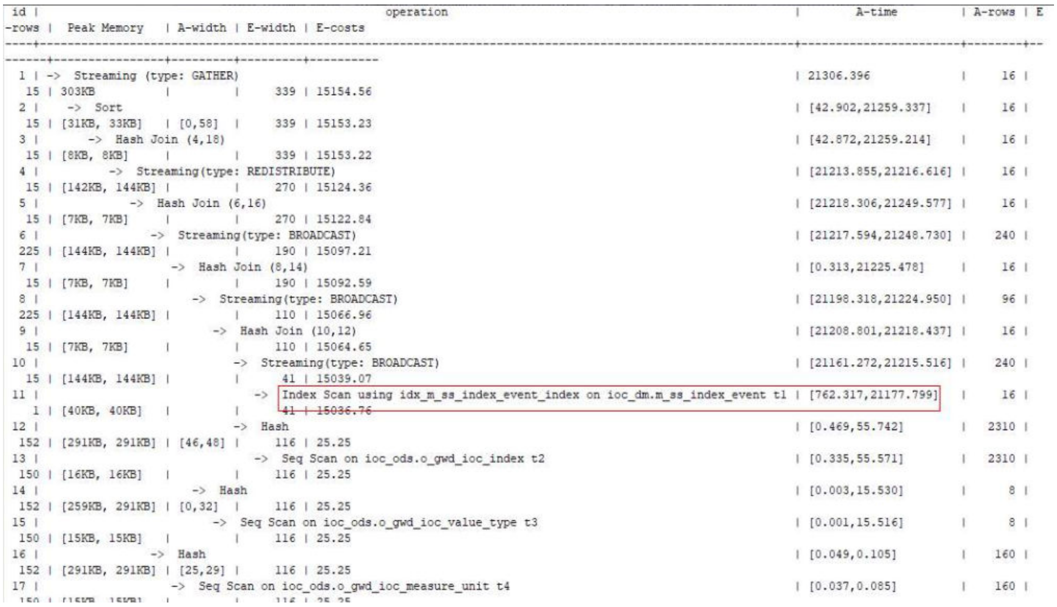
- A large number of tables are frequently added, deleted, and modified and are not cleared in a timely manner. As a result, a large amount of dirty data exists and the table data size expands, resulting in slow query.
- The memory parameters are configured improperly.

Cause Analysis

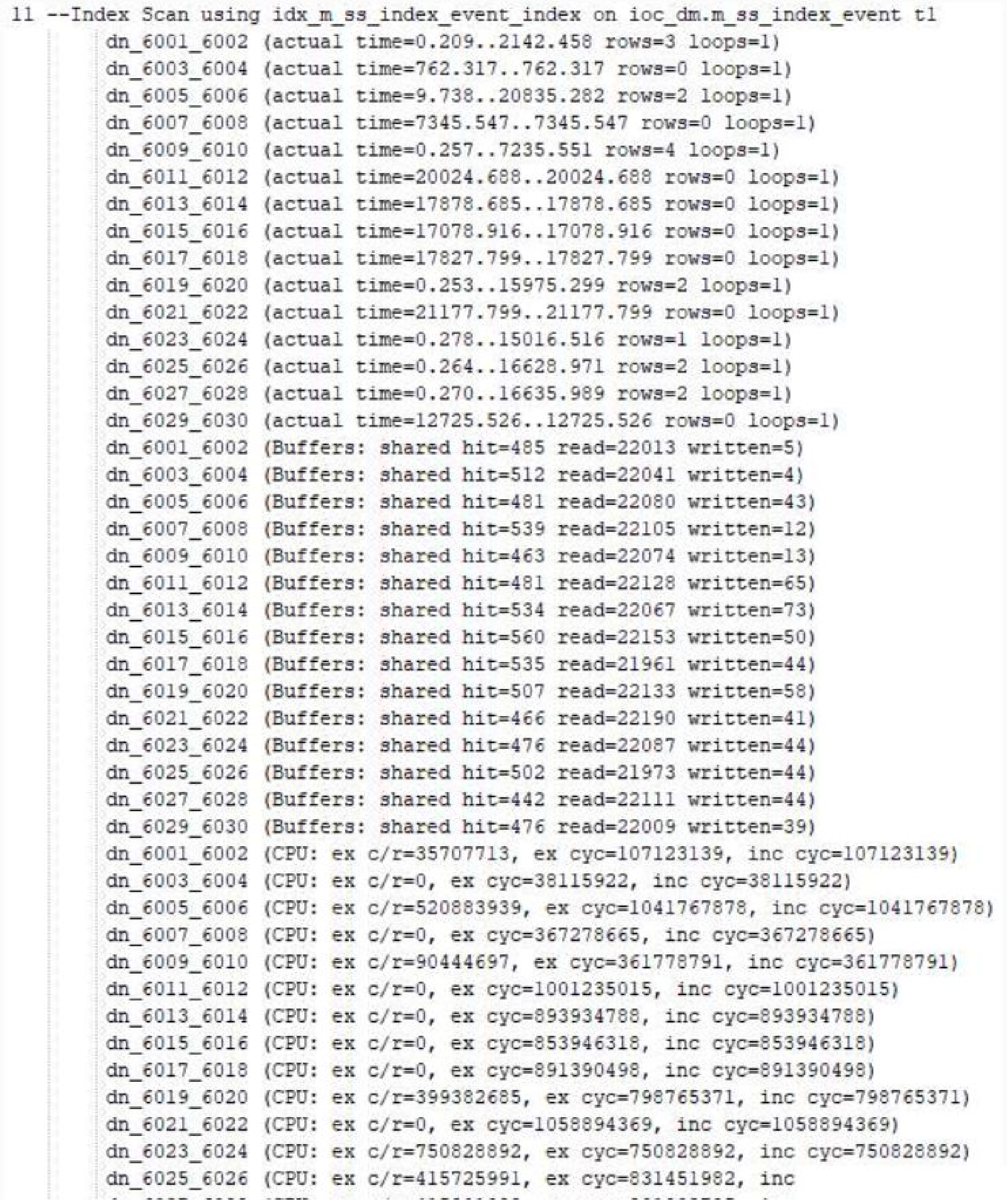
Step 1 The customer confirms that some services are slow. Some slow SQL statements can be provided and the execution plan can be printed. The time is spent mostly on the index scan, which may be caused by I/O contention. After I/O monitoring, no I/O resource usage bottleneck is found.



```
------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------
   3 --Hash Join (4,18)
     |     Hash Cond: ((t1.area_code)::text = (t5.area_code)::text)
   5 --Hash Join (6,16)
     |     Hash Cond: ((t1.measure_unit_code)::text = (t4.measure_unit_code)::text)
   7 --Hash Join (8,14)
     |     Hash Cond: ((t1.value_type_code)::text = (t3.value_type_code)::text)
   9 --Hash Join (10,12)
     |     Hash Cond: ((t1.index_code)::text = (t2.index_code)::text)
  11 --Index Scan using idx_m_ss_index_event_index on ioc_dm.m_ss_index_event t1
     |     Filter: ((t1.data_status = 1::numeric) AND (length((t1.occur_period)::text) = 2) AND ((t1.index_code)::bigint = 100100010011::bigint) AND ((t
1.area_code)::bigint = 440307000000::bigint) AND ((t1.time_type_code)::bigint = 8) AND (to_char(t1.update_time, 'yyyymmdd'::text) = to_char(('2020-02-
25'::date)::timestamp with time zone, 'yyyymmdd'::text)))
     |     Rows Removed by Filter: 51913
(11 rows)
```



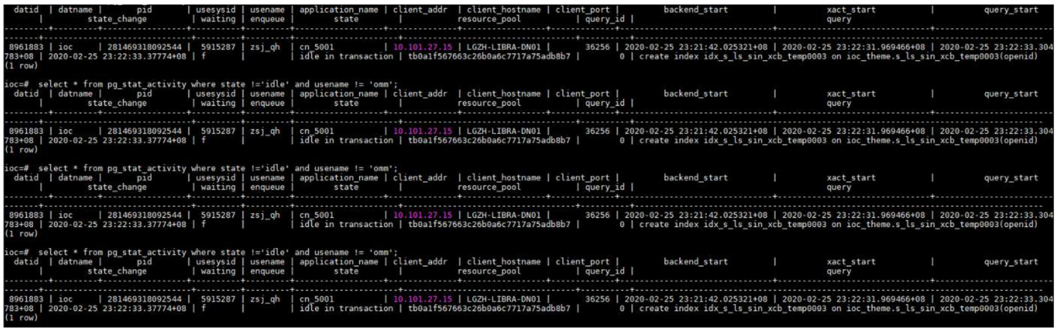
```
 id |                                          operation                                          |        A-time         | A-rows | E
-rows |  Peak Memory   | A-width | E-width | E-costs
----+---------------------------------------------------------------------------------------------+-----------------------+--------+--
------+----------------+---------+---------+----------
  1 | ->  Streaming (type: GATHER)                                                               | 21306.396             |     16 |
  15 | 303KB          |         |     339 | 15154.56
  2 |    ->  Sort                                                                                 | [42.902,21259.337]    |     16 |
  15 | [31KB, 33KB]   | [0,58]  |     339 | 15153.23
  3 |       ->  Hash Join (4,18)                                                                  | [42.872,21259.214]    |     16 |
  15 | [8KB, 8KB]     |         |     339 | 15153.22
  4 |          ->  Streaming(type: REDISTRIBUTE)                                                  | [21213.855,21216.616] |     16 |
  15 | [142KB, 144KB] |         |     270 | 15124.36
  5 |             ->  Hash Join (6,16)                                                            | [21218.306,21249.577] |     16 |
  15 | [7KB, 7KB]     |         |     270 | 15122.84
  6 |                ->  Streaming(type: BROADCAST)                                               | [21217.594,21248.730] |    240 |
 225 | [144KB, 144KB] |         |     190 | 15097.21
  7 |                   ->  Hash Join (8,14)                                                      | [0.313,21225.478]     |     16 |
  15 | [7KB, 7KB]     |         |     190 | 15092.59
  8 |                      ->  Streaming(type: BROADCAST)                                         | [21198.318,21224.950] |     96 |
 225 | [144KB, 144KB] |         |     110 | 15066.96
  9 |                         ->  Hash Join (10,12)                                               | [21208.801,21218.437] |     16 |
  15 | [7KB, 7KB]     |         |     110 | 15064.65
 10 |                            ->  Streaming(type: BROADCAST)                                   | [21161.272,21215.516] |    240 |
  15 | [144KB, 144KB] |         |      41 | 15039.07
 11 |                               ->  Index Scan using idx_m_ss_index_event_index on ioc_dm.m_ss_index_event t1 | [762.317,21177.799]   |     16 |
   1 | [40KB, 40KB]   |         |      41 | 15036.76
 12 |                            ->  Hash                                                         | [0.469,55.742]        |   2310 |
 152 | [291KB, 291KB] | [46,48] |     116 | 25.25
 13 |                               ->  Seq Scan on ioc_ods.o_gwd_ioc_index t2                    | [0.335,55.571]        |   2310 |
 150 | [16KB, 16KB]   |         |     116 | 25.25
 14 |                      ->  Hash                                                               | [0.003,15.530]        |      8 |
 152 | [259KB, 291KB] | [0,32]  |     116 | 25.25
 15 |                         ->  Seq Scan on ioc_ods.o_gwd_ioc_value_type t3                     | [0.001,15.516]        |      8 |
 150 | [15KB, 15KB]   |         |     116 | 25.25
 16 |                ->  Hash                                                                     | [0.049,0.105]         |    160 |
 152 | [291KB, 291KB] | [25,29] |     116 | 25.25
 17 |                   ->  Seq Scan on ioc_ods.o_gwd_ioc_measure_unit t4                         | [0.037,0.085]         |    160 |
 150 | [15KB, 15KB]   |         |     116 | 25.25
```



```
11 --Index Scan using idx_m_ss_index_event_index on ioc_dm.m_ss_index_event t1
      dn_6001_6002 (actual time=0.209..2142.458 rows=3 loops=1)
      dn_6003_6004 (actual time=762.317..762.317 rows=0 loops=1)
      dn_6005_6006 (actual time=9.738..20835.282 rows=2 loops=1)
      dn_6007_6008 (actual time=7345.547..7345.547 rows=0 loops=1)
      dn_6009_6010 (actual time=0.257..7235.551 rows=4 loops=1)
      dn_6011_6012 (actual time=20024.688..20024.688 rows=0 loops=1)
      dn_6013_6014 (actual time=17878.685..17878.685 rows=0 loops=1)
      dn_6015_6016 (actual time=17078.916..17078.916 rows=0 loops=1)
      dn_6017_6018 (actual time=17827.799..17827.799 rows=0 loops=1)
      dn_6019_6020 (actual time=0.253..15975.299 rows=2 loops=1)
      dn_6021_6022 (actual time=21177.799..21177.799 rows=0 loops=1)
      dn_6023_6024 (actual time=0.278..15016.516 rows=1 loops=1)
      dn_6025_6026 (actual time=0.264..16628.971 rows=2 loops=1)
      dn_6027_6028 (actual time=0.270..16635.989 rows=2 loops=1)
      dn_6029_6030 (actual time=12725.526..12725.526 rows=0 loops=1)
      dn_6001_6002 (Buffers: shared hit=485 read=22013 written=5)
      dn_6003_6004 (Buffers: shared hit=512 read=22041 written=4)
      dn_6005_6006 (Buffers: shared hit=481 read=22080 written=43)
      dn_6007_6008 (Buffers: shared hit=539 read=22105 written=12)
      dn_6009_6010 (Buffers: shared hit=463 read=22074 written=13)
      dn_6011_6012 (Buffers: shared hit=481 read=22128 written=65)
      dn_6013_6014 (Buffers: shared hit=534 read=22067 written=73)
      dn_6015_6016 (Buffers: shared hit=560 read=22153 written=50)
      dn_6017_6018 (Buffers: shared hit=535 read=21961 written=44)
      dn_6019_6020 (Buffers: shared hit=507 read=22133 written=58)
      dn_6021_6022 (Buffers: shared hit=466 read=22190 written=41)
      dn_6023_6024 (Buffers: shared hit=476 read=22087 written=44)
      dn_6025_6026 (Buffers: shared hit=502 read=21973 written=44)
      dn_6027_6028 (Buffers: shared hit=442 read=22111 written=44)
      dn_6029_6030 (Buffers: shared hit=476 read=22009 written=39)
      dn_6001_6002 (CPU: ex c/r=35707713, ex cyc=107123139, inc cyc=107123139)
      dn_6003_6004 (CPU: ex c/r=0, ex cyc=38115922, inc cyc=38115922)
      dn_6005_6006 (CPU: ex c/r=520883939, ex cyc=1041767878, inc cyc=1041767878)
      dn_6007_6008 (CPU: ex c/r=0, ex cyc=367278665, inc cyc=367278665)
      dn_6009_6010 (CPU: ex c/r=90444697, ex cyc=361778791, inc cyc=361778791)
      dn_6011_6012 (CPU: ex c/r=0, ex cyc=1001235015, inc cyc=1001235015)
      dn_6013_6014 (CPU: ex c/r=0, ex cyc=893934788, inc cyc=893934788)
      dn_6015_6016 (CPU: ex c/r=0, ex cyc=853946318, inc cyc=853946318)
      dn_6017_6018 (CPU: ex c/r=0, ex cyc=891390498, inc cyc=891390498)
      dn_6019_6020 (CPU: ex c/r=399382685, ex cyc=798765371, inc cyc=798765371)
      dn_6021_6022 (CPU: ex c/r=0, ex cyc=1058894369, inc cyc=1058894369)
      dn_6023_6024 (CPU: ex c/r=750828892, ex cyc=750828892, inc cyc=750828892)
      dn_6025_6026 (CPU: ex c/r=415725991, ex cyc=831451982, inc
```

Step 2 Query the active SQL statements. A large number of **create index** statements are found. Confirm with the customer whether the service is proper.

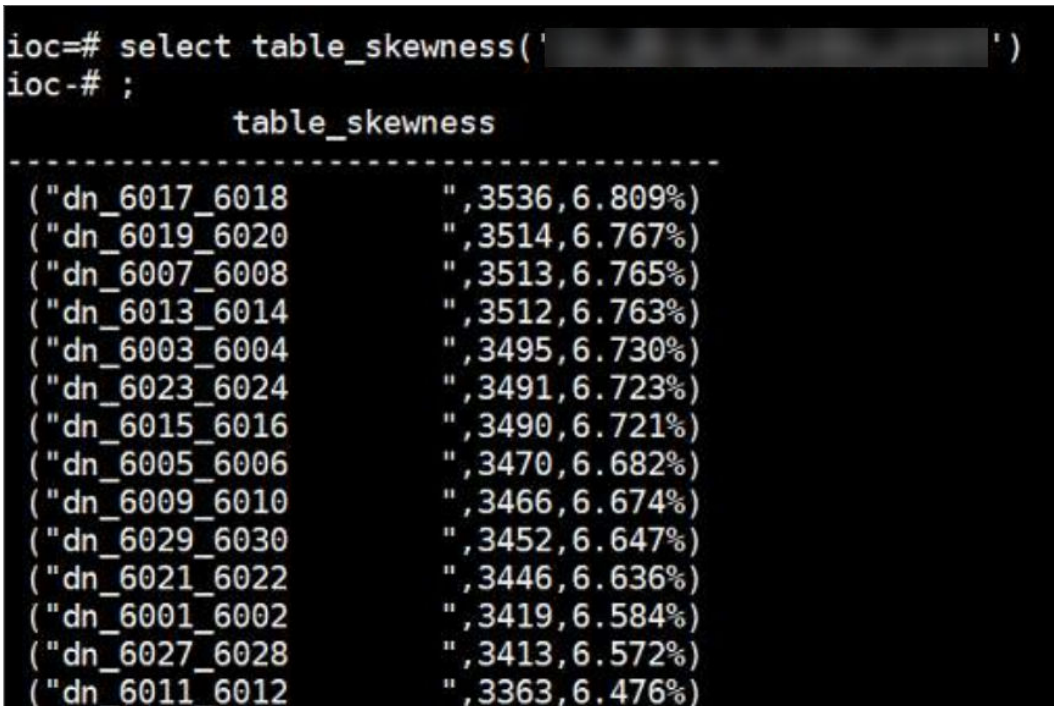
```
SELECT * from pg_stat_activity where state !='idle' and usename !='Ruby';
```



```
  datid  | datname |       pid       | usesysid | usename | application_name | client_addr  | client_hostname | client_port |         backend_start         |          xact_start           |          query_start
         |          state_change          | waiting | enqueue |        state        |                resource_pool            | query_id |                                   query
---------+---------+-----------------+----------+---------+------------------+--------------+-----------------+-------------+-------------------------------+-------------------------------+-------------------------------
---------+--------------------------------+---------+---------+---------------------+-------------------------------------------+----------+-----------------------------------------------------------------------------
 8961883 | ioc     | 281469318092544 |  5915287 | zsj_qh  | cn_5001          | 10.101.27.15 | LGZH-LIBRA-DN01 |       36256 | 2020-02-25 23:21:42.025321+08 | 2020-02-25 23:22:31.969466+08 | 2020-02-25 23:22:33.304
783+08 | 2020-02-25 23:22:33.37774+08 | f        |         | idle in transaction | tb0a1f567663c26b0a6c7717a75adb8b7 |        0 | create index idx_s_ls_sin_xcb_temp0003 on ioc_theme.s_ls_sin_xcb_temp0003(openid)
(1 row)

ioc=#  select * from pg_stat_activity where state !='idle' and usename != 'omm';
  datid  | datname |       pid       | usesysid | usename | application_name | client_addr  | client_hostname | client_port |         backend_start         |          xact_start           |          query_start
         |          state_change          | waiting | enqueue |        state        |                resource_pool            | query_id |                                   query
---------+---------+-----------------+----------+---------+------------------+--------------+-----------------+-------------+-------------------------------+-------------------------------+-------------------------------
---------+--------------------------------+---------+---------+---------------------+-------------------------------------------+----------+-----------------------------------------------------------------------------
 8961883 | ioc     | 281469318092544 |  5915287 | zsj_qh  | cn_5001          | 10.101.27.15 | LGZH-LIBRA-DN01 |       36256 | 2020-02-25 23:21:42.025321+08 | 2020-02-25 23:22:31.969466+08 | 2020-02-25 23:22:33.304
783+08 | 2020-02-25 23:22:33.37774+08 | f        |         | idle in transaction | tb0a1f567663c26b0a6c7717a75adb8b7 |        0 | create index idx_s_ls_sin_xcb_temp0003 on ioc_theme.s_ls_sin_xcb_temp0003(openid)
(1 row)

ioc=#  select * from pg_stat_activity where state !='idle' and usename != 'omm';
  datid  | datname |       pid       | usesysid | usename | application_name | client_addr  | client_hostname | client_port |         backend_start         |          xact_start           |          query_start
         |          state_change          | waiting | enqueue |        state        |                resource_pool            | query_id |                                   query
---------+---------+-----------------+----------+---------+------------------+--------------+-----------------+-------------+-------------------------------+-------------------------------+-------------------------------
---------+--------------------------------+---------+---------+---------------------+-------------------------------------------+----------+-----------------------------------------------------------------------------
 8961883 | ioc     | 281469318092544 |  5915287 | zsj_qh  | cn_5001          | 10.101.27.15 | LGZH-LIBRA-DN01 |       36256 | 2020-02-25 23:21:42.025321+08 | 2020-02-25 23:22:31.969466+08 | 2020-02-25 23:22:33.304
783+08 | 2020-02-25 23:22:33.37774+08 | f        |         | idle in transaction | tb0a1f567663c26b0a6c7717a75adb8b7 |        0 | create index idx_s_ls_sin_xcb_temp0003 on ioc_theme.s_ls_sin_xcb_temp0003(openid)
(1 row)

ioc=#  select * from pg_stat_activity where state !='idle' and usename != 'omm';
  datid  | datname |       pid       | usesysid | usename | application_name | client_addr  | client_hostname | client_port |         backend_start         |          xact_start           |          query_start
         |          state_change          | waiting | enqueue |        state        |                resource_pool            | query_id |                                   query
---------+---------+-----------------+----------+---------+------------------+--------------+-----------------+-------------+-------------------------------+-------------------------------+-------------------------------
---------+--------------------------------+---------+---------+---------------------+-------------------------------------------+----------+-----------------------------------------------------------------------------
 8961883 | ioc     | 281469318092544 |  5915287 | zsj_qh  | cn_5001          | 10.101.27.15 | LGZH-LIBRA-DN01 |       36256 | 2020-02-25 23:21:42.025321+08 | 2020-02-25 23:22:31.969466+08 | 2020-02-25 23:22:33.304
783+08 | 2020-02-25 23:22:33.37774+08 | f        |         | idle in transaction | tb0a1f567663c26b0a6c7717a75adb8b7 |        0 | create index idx_s_ls_sin_xcb_temp0003 on ioc_theme.s_ls_sin_xcb_temp0003(openid)
(1 row)
```

Step 3 According to the execution plan, it takes a long time to execute statements on some DNs. No table skew occurs.

```
SELECT table_skewness('table name');
```



```
ioc=# select table_skewness('[redacted]')
ioc-# ;
          table_skewness
-------------------------------------
 ("dn_6017_6018       ",3536,6.809%)
 ("dn_6019_6020       ",3514,6.767%)
 ("dn_6007_6008       ",3513,6.765%)
 ("dn_6013_6014       ",3512,6.763%)
 ("dn_6003_6004       ",3495,6.730%)
 ("dn_6023_6024       ",3491,6.723%)
 ("dn_6015_6016       ",3490,6.721%)
 ("dn_6005_6006       ",3470,6.682%)
 ("dn_6009_6010       ",3466,6.674%)
 ("dn_6029_6030       ",3452,6.647%)
 ("dn_6021_6022       ",3446,6.636%)
 ("dn_6001_6002       ",3419,6.584%)
 ("dn_6027_6028       ",3413,6.572%)
 ("dn_6011_6012       ",3363,6.476%)
```

Step 4 Contact the O&M personnel to log in to the cluster instance and check the memory parameters. The parameters are configured improperly and need to be adjusted.

- The total memory size of a single node is 256 GB.
- The value of **max_process_memory** is **12 GB**, which is too small.
- The value of **shared_buffers** is **32 MB**, which is too small.
- **work_mem**: **CN**: **64 MB**; **DN**: **64 MB**
- The value of **max_active_statements** is **-1**. (The number of concurrent statements is not limited.)

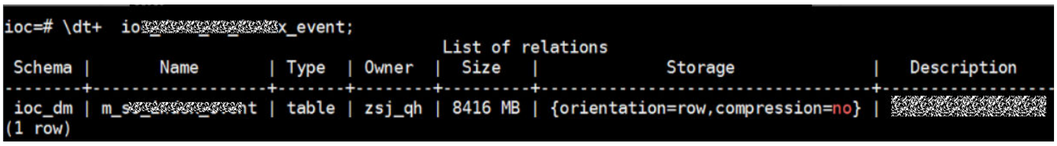
Step 5 Configure this parameter as follows:

gs_guc set -Z coordinator -Z datanode -N all -I all -c "max_process_memory=25GB"

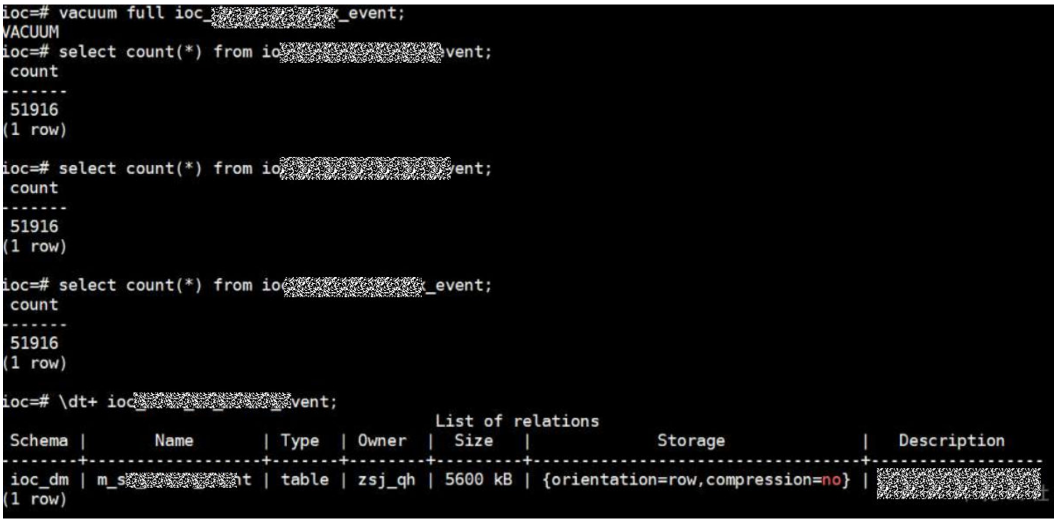
gs_guc set -Z coordinator -Z datanode -N all -I all -c "shared_buffers=8GB"

gs_guc set -Z coordinator -Z datanode -N all -I all -c "work_mem=128MB"

Step 6 It is found that the table data size expands excessively. After the **VACUUM FULL** operation is performed on an 8 GB table that contains 50,000 data records, the table size decreases to 5.6 MB.



```
ioc=# \dt+  io[redacted]x_event;
                                        List of relations
 Schema |       Name       | Type  | Owner  |  Size   |              Storage               |  Description
--------+------------------+-------+--------+---------+------------------------------------+-----------------
 ioc_dm | m_s[redacted]nt | table | zsj_qh | 8416 MB | {orientation=row,compression=no} | [redacted]
(1 row)
```



```
ioc=# vacuum full ioc_[redacted]_event;
VACUUM
ioc=# select count(*) from io[redacted]vent;
 count
-------
 51916
(1 row)

ioc=# select count(*) from io[redacted]vent;
 count
-------
 51916
(1 row)

ioc=# select count(*) from io[redacted]_event;
 count
-------
 51916
(1 row)

ioc=# \dt+ ioc[redacted]vent;
                                            List of relations
 Schema |       Name       | Type  | Owner  |  Size   |              Storage              |   Description
--------+------------------+-------+--------+---------+-----------------------------------+------------------
 ioc_dm | m_s[redacted]nt | table | zsj_qh | 5600 kB | {orientation=row,compression=no} | [redacted]
(1 row)
```

----End

Handling Procedure

Step 1 Perform the **VACUUM FULL** operation on large tables to clear dirty data.

Step 2 Configure GUC memory parameters.

----End

6.12 Memory Overflow Occurs in a Cluster

Symptom

The error log is as follows:

```
[ERROR] Mpp task queryDataAnalyseById or updateDataAnalyseHistoryEndTimesAndResult fail,
dataAnalyseId:17615 org.postgresql.util.PSQLException: ERROR: memory is temporarily unavailable
sql: vacuum full dws_customer_360.t_user_resource;
```

Possible Causes

Some SQL statements have exhausted memory. When other statements are executed, no memory can be allocated, and a message is displayed indicating that the memory is insufficient.

Handling Procedure

1. Adjust the service execution time window to ensure that the service execution time is different from the time when a large number of concurrent services are executed.
2. Query the memory usage of the current cluster, find the statements with high memory usage, and terminate them to release the cluster memory. Here is the procedure:

- **For a cluster of 8.1.1 or an earlier version, perform the following steps:**

Step 1 Run the following statement to query the memory usage of the current cluster and check whether the value of **dynamic_used_memory** of an instance is greater than or close to the value of **max_dynamic_memory**. If the preceding error is reported, the value of **dynamic_used_memory** reaches the upper limit.

```
SELECT * FROM pgxc_total_memory_detail;
```

Step 2 When the Top SQL feature is enabled, run the real-time **TOP SQL** command to query the query statements that use most of the memory. You can find the statements that consume a large amount of memory based on the values of **max_peak_memory** and **memory_skew_percent** in the command output.

```
SELECT
nodename,pid,dbname,username,application_name,min_peak_memory,max_peak_memory,average_peak_me
mory,memory_skew_percent,substr(query,0,50) as query FROM pgxc_wlm_session_statistics;
```

Step 3 Based on the session information obtained in **Step 2**, execute the **pg_terminate_backend** function to end the corresponding session to restore the memory. After the restoration, you can optimize the SQL statements that consume large memory.

```
SELECT pg_terminate_backend(pid);
```

----End

- **For clusters of 8.1.2 or later, you can log in to the GaussDB(DWS) management console and perform the following steps on the real-time query monitoring page:**

NOTICE

- Real-time query is supported only in clusters of version 8.1.2 and later.
- To enable the real-time query function, choose **Monitoring** > **Monitoring Collection** and enable **Real-Time Query Monitoring**.

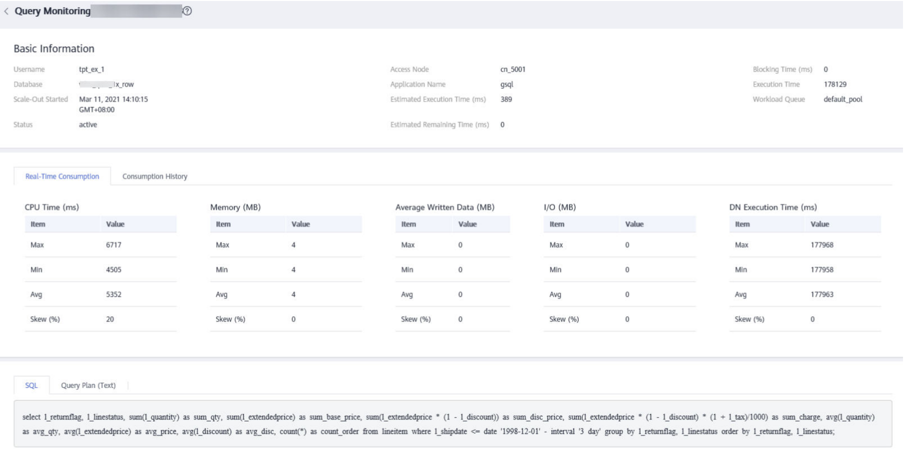
Step 1 Log in to the GaussDB(DWS) management console. On the **Clusters** page, locate the target cluster and click **Monitoring Panel** in the **Operation** column. The database monitoring page is displayed.

Step 2 In the navigation pane, choose **Monitoring** > **Queries**.

Step 3 Choose a time period and view queries executed in the cluster.

Step 4 Click a session query ID to view the monitoring details. The detail information includes the **Username**, **Database Name**, **Execution Time**, **Query Statement**, **Query Status**, **Workload Queue**, **Min. Peak DN Memory**, **Max. Peak DN Memory**, **Max. Peak IOPS on DN**, **Min. Peak IOPS on DN**, and **Average Memory Usage**.

A larger value of **Max. Peak DN Memory** or **Average Memory Usage** indicates a larger memory usage.



Basic Information

Username	tpt_ex_1	Access Node	cn_5001	Blocking Time (ms)	0
Database	[redacted]x_row	Application Name	gsql	Execution Time	178129
Scale-Out Started	Mar 11, 2021 14:10:15 GMT+08:00	Estimated Execution Time (ms)	389	Workload Queue	default_pool
Status	active	Estimated Remaining Time (ms)	0		

CPU Time (ms)

Item	Value
Max	6717
Min	4505
Avg	5352
Skew (%)	20

Memory (MB)

Item	Value
Max	4
Min	4
Avg	4
Skew (%)	0

Average Written Data (MB)

Item	Value
Max	0
Min	0
Avg	0
Skew (%)	0

I/O (MB)

Item	Value
Max	0
Min	0
Avg	0
Skew (%)	0

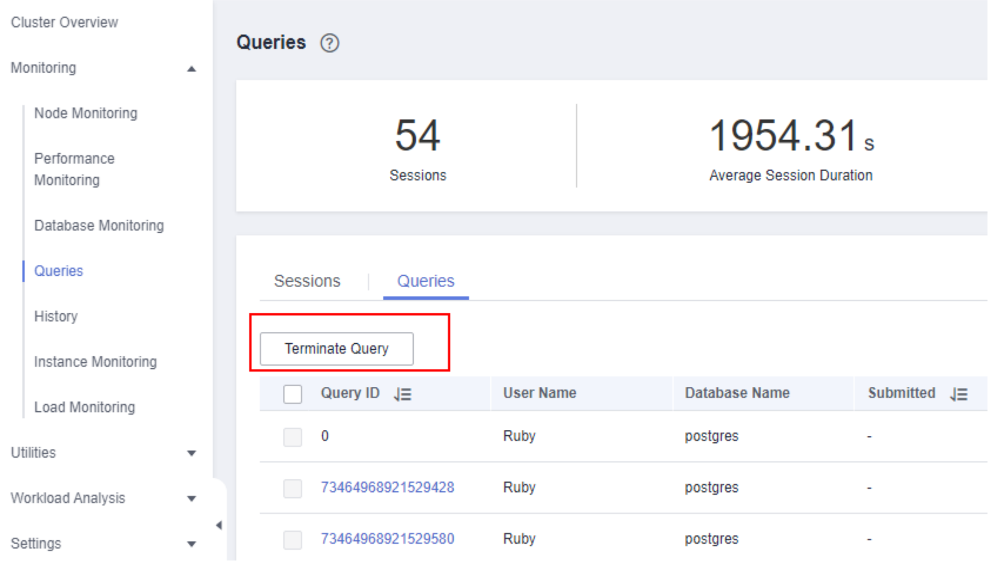
DN Execution Time (ms)

Item	Value
Max	177968
Min	177958
Avg	177963
Skew (%)	0

```
select l_returnflag, l_linestatus, sum(l_quantity) as sum_qty, sum(l_extendedprice) as sum_base_price, sum(l_extendedprice * (1 - l_discount)) as sum_disc_price, sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)/1000) as sum_charge, avg(l_quantity)
as avg_qty, avg(l_extendedprice) as avg_price, avg(l_discount) as avg_disc, count(*) as count_order from lineitem where l_shipdate <= date '1998-12-01' - interval '3 day' group by l_returnflag, l_linestatus order by l_returnflag, l_linestatus;
```

Step 5 If you have confirmed that a statement with high memory usage needs to be terminated, select the query ID and click **Terminate Query** to terminate the query.

The fine-grained permission control function is added. Only users with the operate permission are able to terminate queries. For users with the read-only permission, the **Terminate Query** button is grayed out.



□	Query ID	User Name	Database Name	Submitted
□	0	Ruby	postgres	-
□	73464968921529428	Ruby	postgres	-
□	73464968921529580	Ruby	postgres	-

----End

6.13 Statements with User-defined Functions Cannot Be Pushed Down

Symptom

SQL statements cannot be pushed down.

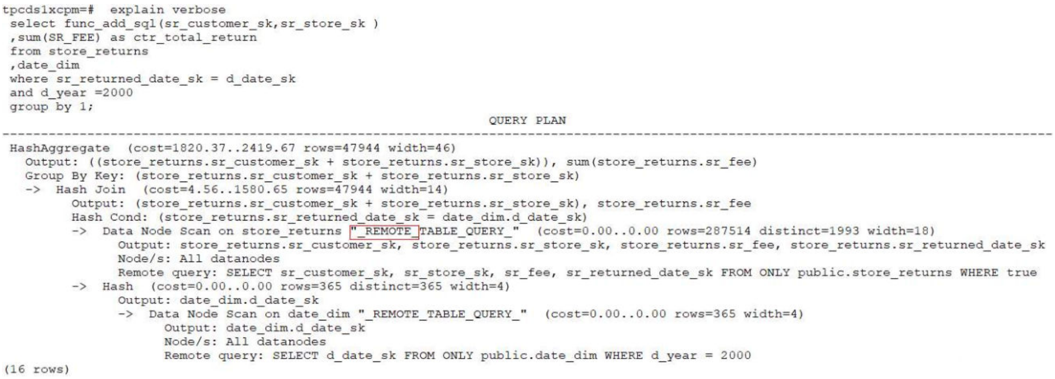
Possible Causes

The latest version supports the pushdown of most common functions. Statements cannot be pushed down mainly because the attributes of user-defined functions are incorrectly defined.

If statements are not pushed down, the advantages of distributed computing are not leveraged. In this case, massive data is processed only by one node during the statement execution, resulting in poor performance.

Cause Analysis

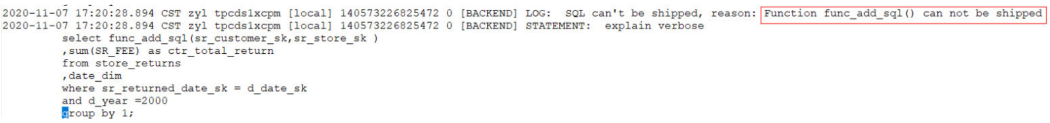
Step 1 Run **explain verbose** to print the execution plan of a statement.



```
tpcds1xcpm=#  explain verbose
 select func_add_sql(sr_customer_sk,sr_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 1;
                                                              QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------
 HashAggregate  (cost=1820.37..2419.67 rows=47944 width=46)
   Output: ((store_returns.sr_customer_sk + store_returns.sr_store_sk)), sum(store_returns.sr_fee)
   Group By Key: (store_returns.sr_customer_sk + store_returns.sr_store_sk)
   ->  Hash Join  (cost=4.56..1580.65 rows=47944 width=14)
         Output: (store_returns.sr_customer_sk + store_returns.sr_store_sk), store_returns.sr_fee
         Hash Cond: (store_returns.sr_returned_date_sk = date_dim.d_date_sk)
         ->  Data Node Scan on store_returns "_REMOTE_TABLE_QUERY_"  (cost=0.00..0.00 rows=287514 distinct=1993 width=18)
               Output: store_returns.sr_customer_sk, store_returns.sr_store_sk, store_returns.sr_fee, store_returns.sr_returned_date_sk
               Node/s: All datanodes
               Remote query: SELECT sr_customer_sk, sr_store_sk, sr_fee, sr_returned_date_sk FROM ONLY public.store_returns WHERE true
         ->  Hash  (cost=0.00..0.00 rows=365 distinct=365 width=4)
               Output: date_dim.d_date_sk
               ->  Data Node Scan on date_dim "_REMOTE_TABLE_QUERY_"  (cost=0.00..0.00 rows=365 width=4)
                     Output: date_dim.d_date_sk
                     Node/s: All datanodes
                     Remote query: SELECT d_date_sk FROM ONLY public.date_dim WHERE d_year = 2000
(16 rows)
```

The **__REMOTE** keyword in the preceding execution plan indicates that the current statement cannot be pushed down.

Step 2 The reason why a statement cannot be pushed down is printed in **pg_log**. The CN logs of the preceding statement are similar to the following.



```
2020-11-07 17:20:28.894 CST zyl tpcds1xcpm [local] 140573226825472 0 [BACKEND] LOG:  SQL can't be shipped, reason: Function func_add_sql() can not be shipped
2020-11-07 17:20:28.894 CST zyl tpcds1xcpm [local] 140573226825472 0 [BACKEND] STATEMENT:  explain verbose
        select func_add_sql(sr_customer_sk,sr_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 1;
```

----End

Handling Procedure

Check whether the **provolatile** attribute of the user-defined function is correctly defined. If the definition is incorrect, modify the corresponding attribute so that the statement can be pushed down.

For details, see the following description.

- All attributes related to the function can be queried in the **pg_proc** system catalog. The two attributes that determine whether the function can be pushed down are **provolatile** and **proshippable**.
  - If the **provolatile** of a function is **i**, the function can be pushed down regardless of the value of **proshippable**.
  - If the **provolatile** of a function is **s** or **v**, the function can be pushed only if the value of **proshippable** is **t**.

- **provolatile** is to describe the volatile attribute of a function. The value can be **i**, **s**, or **v**. **i** indicates **IMMUTABLE**, **s** indicates **STABLE**, and **v** indicates **VOLATILE**.

  Examples are as follows:

  – If a function must have the same output for the same input, this function is **IMMUTABLE**, such as most string processing functions. These functions can always be pushed down.

  – If the returned result of a function is the same during the calling of an SQL statement, the function is **STABLE**. For example, the final displayed result of time-related processing functions may vary with specific GUC parameters, such as the parameter that determines the time display format. These functions can be pushed down only when their attributes are **SHIPPABLE**.

  – If the returned result of a function varies with each call, the function is **VOLATILE**. For example, the results of invoking the **nextval** and **random** functions are unpredictable. These functions can be pushed down only when their attributes are **SHIPPABLE**.

- **proshippable** indicates whether a function can be pushed down to DNs. The default value is **false**, and the value can be **true**, **false**, or **NULL**.

6.14 Column-Store Tables Cannot Be Updated or Table Bloat Occurs

Symptom

- The column-store table fails to be updated.
- When a column-store table is updated for multiple times, the table size is expanded by more than 10 times.

Possible Causes

- Column-store tables cannot be updated concurrently.
- When a column-store table is updated, the space does not reclaim old records.

Handling Procedure

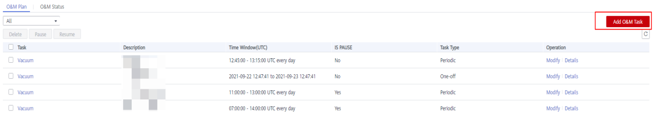
- **Method 1**

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

Step 2 Click the name of the target cluster.

Step 3 In the navigation pane, choose **Intelligent O&M**.

Step 4 Click the **O&M Plan** tab. Click **Add O&M Task**.

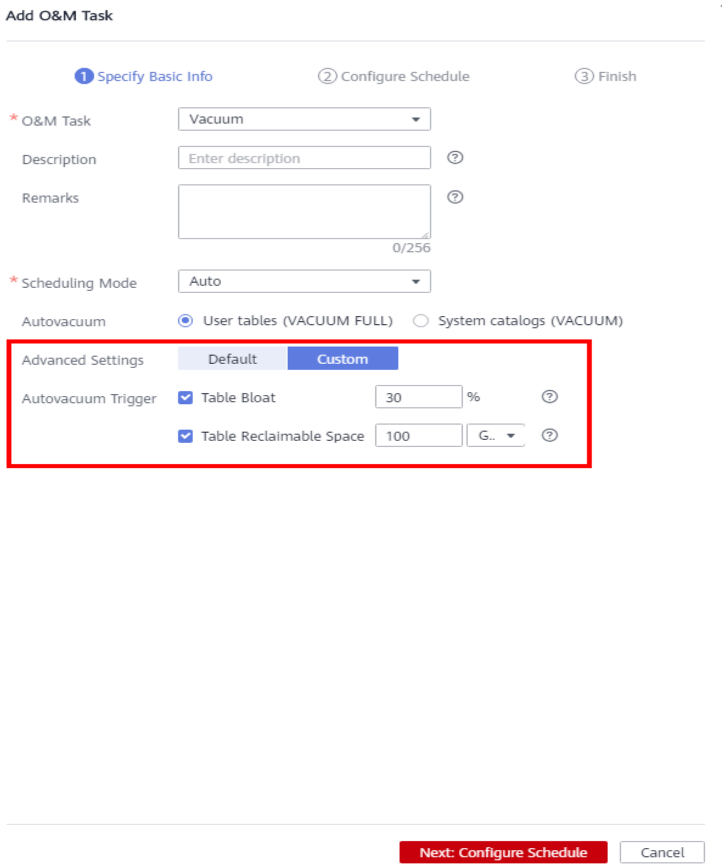


Task	Description	Time Window(UTC)	IS PAUSE	Task Type	Operation
Vacuum	[redacted]	12:45:00 - 13:15:00 UTC every day	No	Periodic	Modify | Details
Vacuum	[redacted]	2021-09-22 12:47:41 to 2021-09-23 12:47:41	No	One-off	Modify | Details
Vacuum	[redacted]	11:00:00 - 13:00:00 UTC every day	Yes	Periodic	Modify | Details
Vacuum	[redacted]	07:00:00 - 14:00:00 UTC every day	Yes	Periodic	Modify | Details

Step 5 The **Add O&M Task** page is displayed.

- Select a **Vacuum** task.
- Select **Specify** for **Scheduling Mode**. The intelligent O&M module automatically delivers table-level Vacuum tasks in the specified time window.

  You can enter column-store tables that require **Vacuum**. Each row corresponds to a table, which contains database name, schema name, and table name, separated by spaces.



Add O&M Task

1 Specify Basic Info 2 Configure Schedule 3 Finish

* O&M Task: Vacuum

Description: Enter description

Remarks: 0/256

* Scheduling Mode: Auto

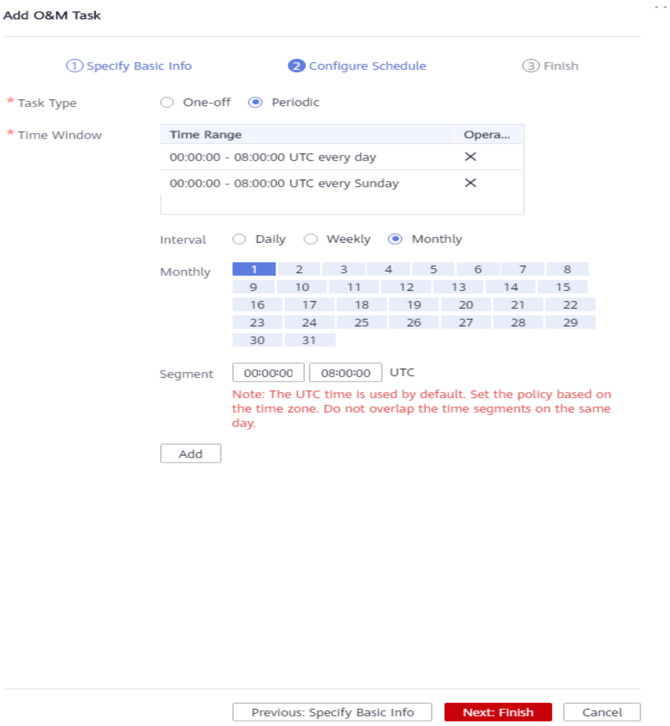
Autovacuum: ● User tables (VACUUM FULL) ○ System catalogs (VACUUM)

Advanced Settings: Default [✓ Custom]

Autovacuum Trigger: ☑ Table Bloat 30 %

☑ Table Reclaimable Space 100 G..

Step 6 Click **Next: Configure Schedule** to configure the schedule and Vacuum type. You are advised to select **Periodic** for **Task Type**. The GaussDB(DWS) automatically executes VACUUM in your selected time windows.



Add O&M Task

① Specify Basic Info ② Configure Schedule ③ Finish

* Task Type ○ One-off ● Periodic

* Time Window

Time Range	Opera...
00:00:00 - 08:00:00 UTC every day	×
00:00:00 - 08:00:00 UTC every Sunday	×

Interval ○ Daily ○ Weekly ● Monthly

Monthly [✓ 1] 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

Segment 00:00:00 08:00:00 UTC

Note: The UTC time is used by default. Set the policy based on the time zone. Do not overlap the time segments on the same day.

Step 7 After confirming that the information is correct, click **Next** to complete the configuration.

----End

- **Method 2**

After updating column-store tables, you perform **VACUUM FULL** to clear the tables. For details, see "VACUUM" in .

```
VACUUM FULL table_name;
```

6.15 Table Bloat Occurs After Data Is Inserted into a Column-Store Table for Multiple Times

Symptom

After INSERT is executed for multiple times in a column-store table, table bloat occurs.

Possible Causes

In a column-store table, data is stored by column. Every 60,000 rows in a column are stored as a CU. CUs in the same column are continuously stored in a file. When the file is larger than 1 GB, more CUs will be stored in a new file. Data in a CU file cannot be modified and can only be appended. After VACUUM is performed on a column-store table that is frequently deleted and updated, even the space marked as available cannot be reused because the data in a CU file cannot be changed. To reuse the space, you need to change the CUs. Therefore, you are not advised to frequently delete and update column-store tables in GaussDB(DWS).

Handling Procedure

You are advised to enable the delta table function for column-store tables.

```
ALTER TABLE table_name SET (ENABLE_DELTA = ON);
```

NOTE

- Enabling the delta table function of a column-store table can prevent small CUs from being generated when a single piece of data or a small amount of data is imported to the table, hence improving performance. For example, if 100 pieces of data are imported each time in a cluster with 3 CNs and 6 DNs, the import time can be reduced by 25%, the storage space usage can be reduced by 97%. Therefore, you need to enable the delta table before inserting a small batch of data for multiple times and disable the delta table after confirming that no small batch of data needs to be imported.
- A delta table is a row-store table attached to a column-store table. After data is inserted into a delta table, the high compression ratio of the column-store table is lost. In normal cases, column-store tables are used to import a large amount of data. Therefore, the delta table is disabled by default, if the delta table is enabled when a large amount of data is imported, more time and space are consumed. If the delta table is enabled when 10,000 data records each time are imported in a cluster with 3 DNs and 6 DNs, the import speed is four times slower and more than 10 times of the space is consumed than that when the delta table is disabled. Therefore, exercise caution when enabling the delta table.

6.16 Writing Data to GaussDB(DWS) Is Slow and Client Data Is Stacked

Symptom

Writing data to GaussDB(DWS) is slow and the client data is stacked.

Possible Causes

If a single **INSERT INTO** statement is used to write data to a database, the client may encounter a bottleneck. **INSERT** is the simplest data writing method and is applicable to scenarios with small data volumes and low concurrency.

Handling Procedure

If data writing is slow, use either of the following methods to rectify the fault:

- Select another more efficient data import mode, for example, **COPY**.

  For details about the import modes, see **Import Modes**.
- Increase the concurrent client requests.

6.17 Low Query Efficiency

A query task that used to take a few milliseconds to complete is now requiring several seconds, and that used to take several seconds is now requiring even half an hour. This section describes how to analyze and rectify such low efficiency issues.

Solution

Perform the following procedure to locate the cause of this fault.

Step 1 Run the **analyze** command to analyze the database.

The **analyze** command updates data statistics information, such as data sizes and attributes in all tables. This is a lightweight command and can be executed frequently. If the query efficiency is improved or restored after the command execution, the autovacuum process does not function well and requires further analysis.

Step 2 Check whether the query statement returns unnecessary information.

For example, if the query statement to query all records in a table, and use result in only the first 10 records. If the ACS contains 50 record table. The query up is very fast. However, when the table contains records reaches 50000 to query efficiency will decrease.

If an application requires only a part of data information but the query statement returns all information, add a LIMIT clause to the query statement to restrict the number of returned records. In this way, the database optimizer can optimize space and improve query efficiency.

Step 3 Check whether the query statement still has a low response even when it is solely executed.

Run the query statement when there are no or only a few other query requests in the database, and observe the query efficiency. If the efficiency is high, the previous issue is possibly caused by a heavily loaded host in the database system or an inefficient execution plan.

Step 4 Check the same query statement repeatedly to check the query efficiency.

One major cause of low query efficiency is that the required information is not cached in the memory or is replaced by other query requests because of insufficient memory resources.

Run the same query statement repeatedly. If the query efficiency increases gradually, the previous issue might be caused by this reason.

----End

6.18 Poor Query Performance Due to the Lack of Statistics

Symptom

The SQL query performance is poor. Warning information is displayed when **EXPLAIN VERBOSE** is executed.

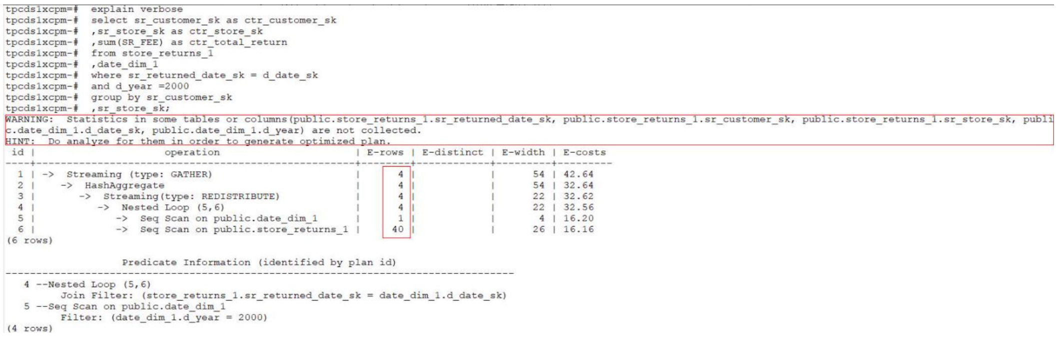
Possible Causes

Statistics about the tables or columns involved in the query are not collected. Without statistics, the execution plan generated by the optimizer will be ineffective, and various performance issues may occur, such as nested loop for equi-join, large table broadcast, and continuous increase of cluster CPU usage.

Cause Analysis

Step 1 Run **explain verbose/explain performance** to print the execution plan of a statement.

The alarm indicating that statistics are not collected for some statements exists in the execution plan, and the estimated value of **E-rows** is small.



```
tpcds1xcpm=#  explain verbose
tpcds1xcpm-#  select sr_customer_sk as ctr_customer_sk
tpcds1xcpm-#  ,sr_store_sk as ctr_store_sk
tpcds1xcpm-#  ,sum(SR_FEE) as ctr_total_return
tpcds1xcpm-#  from store_returns_1
tpcds1xcpm-#  ,date_dim_1
tpcds1xcpm-#  where sr_returned_date_sk = d_date_sk
tpcds1xcpm-#  and d_year =2000
tpcds1xcpm-#  group by sr_customer_sk
tpcds1xcpm-#  ,sr_store_sk;
WARNING:  Statistics in some tables or columns(public.store_returns_1.sr_returned_date_sk, public.store_returns_1.sr_customer_sk, public.store_returns_1.sr_store_sk, publi
c.date_dim_1.d_date_sk, public.date_dim_1.d_year) are not collected.
HINT:  Do analyze for them in order to generate optimized plan.
 id |                      operation                     | E-rows | E-distinct | E-width | E-costs
----+----------------------------------------------------+--------+------------+---------+---------
  1 | ->  Streaming (type: GATHER)                       |      4 |            |      54 | 42.64
  2 |    ->  HashAggregate                               |      4 |            |      54 | 32.64
  3 |       ->  Streaming(type: REDISTRIBUTE)            |      4 |            |      22 | 32.62
  4 |          ->  Nested Loop (5,6)                     |      4 |            |      22 | 32.56
  5 |             ->  Seq Scan on public.date_dim_1      |      1 |            |       4 | 16.20
  6 |             ->  Seq Scan on public.store_returns_1 |     40 |            |      26 | 16.16
(6 rows)

                   Predicate Information (identified by plan id)
----------------------------------------------------------------------------------
   4 --Nested Loop (5,6)
         Join Filter: (store_returns_1.sr_returned_date_sk = date_dim_1.d_date_sk)
   5 --Seq Scan on public.date_dim_1
         Filter: (date_dim_1.d_year = 2000)
(4 rows)
```

Step 2 In the preceding example, a warning message is displayed in the printed execution plan, indicating which columns that are used in the execution plan do not contain statistics.

Similar warning messages are displayed in the **pg_log** log of the CN, and the value of **E-rows** is much smaller than the actual value.

----End

Handling Procedure

You are advised to periodically execute **ANALYZE** or execute it immediately after most of the table data is updated.

6.19 Execution of SQL Statements with NOT IN and NOT EXISTS Is Slow Due to Nested Loops in Execution Plans

Symptom

The customer's SQL statement execution is slow, and the execution plan contains nested loops.

Possible Causes

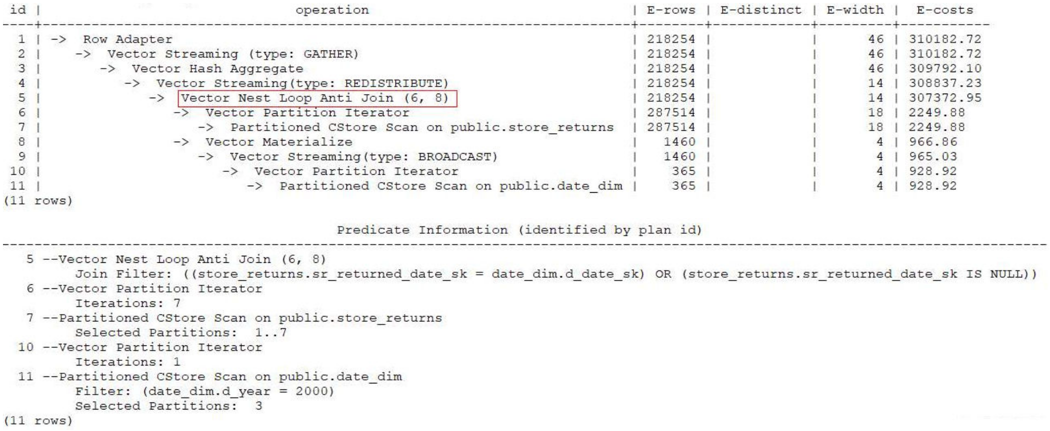
- Nested loop is the main cause of slow statement execution.
- Hash joins can be used only for equi-join, and the nested loop conditions contain **OR**. Therefore, hash joins cannot be used to solve the problem.
- This occurs due to the **NOT IN** syntax. For details, see the description of **NOT IN** and **NOT EXISTS** on the Internet.

Cause Analysis

Step 1 Run **explain verbose** to print the statement execution plan and check whether the SQL statement contains the **NOT IN** syntax.

```
explain verbose
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
where sr_returned_date_sk not in ( select d_date_sk from date_dim where d_year = 2000)
group by sr_customer_sk
,sr_store_sk;
```

Step 2 Nested loop exists in the execution plan.

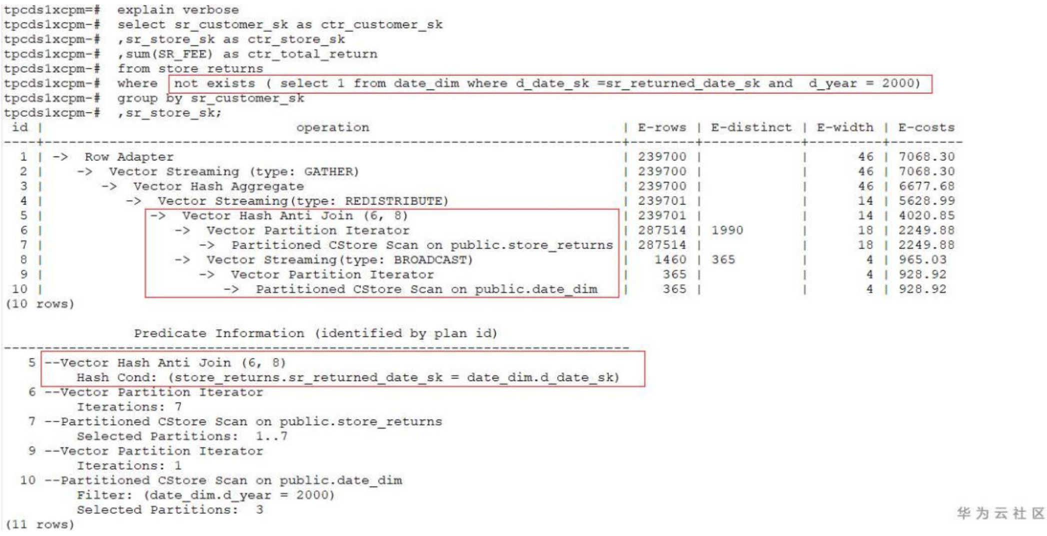


```
 id |                          operation                                  | E-rows | E-distinct | E-width |  E-costs
----+---------------------------------------------------------------------+--------+------------+---------+-----------
  1 | ->  Row Adapter                                                     | 218254 |            |      46 | 310182.72
  2 |    ->  Vector Streaming (type: GATHER)                              | 218254 |            |      46 | 310182.72
  3 |       ->  Vector Hash Aggregate                                     | 218254 |            |      46 | 309792.10
  4 |          ->  Vector Streaming(type: REDISTRIBUTE)                   | 218254 |            |      14 | 308837.23
  5 |             ->  Vector Nest Loop Anti Join (6, 8)                   | 218254 |            |      14 | 307372.95
  6 |                ->  Vector Partition Iterator                        | 287514 |            |      18 | 2249.88
  7 |                   ->  Partitioned CStore Scan on public.store_returns | 287514 |          |      18 | 2249.88
  8 |                ->  Vector Materialize                               |   1460 |            |       4 | 966.86
  9 |                   ->  Vector Streaming(type: BROADCAST)             |   1460 |            |       4 | 965.03
 10 |                      ->  Vector Partition Iterator                  |    365 |            |       4 | 928.92
 11 |                         ->  Partitioned CStore Scan on public.date_dim |  365 |          |       4 | 928.92
(11 rows)

                            Predicate Information (identified by plan id)
-----------------------------------------------------------------------------------------------------------------
   5 --Vector Nest Loop Anti Join (6, 8)
         Join Filter: ((store_returns.sr_returned_date_sk = date_dim.d_date_sk) OR (store_returns.sr_returned_date_sk IS NULL))
   6 --Vector Partition Iterator
         Iterations: 7
   7 --Partitioned CStore Scan on public.store_returns
         Selected Partitions:  1..7
  10 --Vector Partition Iterator
         Iterations: 1
  11 --Partitioned CStore Scan on public.date_dim
         Filter: (date_dim.d_year = 2000)
         Selected Partitions:  3
(11 rows)
```

----End

Handling Procedure

Step 1 In most scenarios, the required result set can be obtained using **NOT EXISTS**. Therefore, you can change **NOT IN** to **NOT EXISTS** in the preceding statement.



```
tpcds1xcpm=#  explain verbose
tpcds1xcpm-#  select sr_customer_sk as ctr_customer_sk
tpcds1xcpm-#  ,sr_store_sk as ctr_store_sk
tpcds1xcpm-#  ,sum(SR_FEE) as ctr_total_return
tpcds1xcpm-#  from store_returns
tpcds1xcpm-#  where  not exists ( select 1 from date_dim where d_date_sk =sr_returned_date_sk and  d_year = 2000)
tpcds1xcpm-#  group by sr_customer_sk
tpcds1xcpm-#  ,sr_store_sk;
 id |                          operation                                  | E-rows | E-distinct | E-width | E-costs
----+---------------------------------------------------------------------+--------+------------+---------+---------
  1 | ->  Row Adapter                                                     | 239700 |            |      46 | 7068.30
  2 |    ->  Vector Streaming (type: GATHER)                              | 239700 |            |      46 | 7068.30
  3 |       ->  Vector Hash Aggregate                                     | 239700 |            |      46 | 6677.68
  4 |          ->  Vector Streaming(type: REDISTRIBUTE)                   | 239701 |            |      14 | 5628.99
  5 |             ->  Vector Hash Anti Join (6, 8)                        | 239701 |            |      14 | 4020.85
  6 |                ->  Vector Partition Iterator                        | 287514 | 1990       |      18 | 2249.88
  7 |                   ->  Partitioned CStore Scan on public.store_returns | 287514 |          |      18 | 2249.88
  8 |                ->  Vector Streaming(type: BROADCAST)                |   1460 | 365        |       4 | 965.03
  9 |                   ->  Vector Partition Iterator                     |    365 |            |       4 | 928.92
 10 |                      ->  Partitioned CStore Scan on public.date_dim |    365 |            |       4 | 928.92
(10 rows)

                  Predicate Information (identified by plan id)
-----------------------------------------------------------------------------
   5 --Vector Hash Anti Join (6, 8)
         Hash Cond: (store_returns.sr_returned_date_sk = date_dim.d_date_sk)
   6 --Vector Partition Iterator
         Iterations: 7
   7 --Partitioned CStore Scan on public.store_returns
         Selected Partitions:  1..7
   9 --Vector Partition Iterator
         Iterations: 1
  10 --Partitioned CStore Scan on public.date_dim
         Filter: (date_dim.d_year = 2000)
         Selected Partitions:  3
(11 rows)
```

----End

6.20 SQL Query Is Slow Because Partitions Are Not Pruned

Symptom

The query of three SQL statements is slow. The queried partitioned table contains 18.5 billion data records, and the query criteria do not contain the partition key.

```
SELECT passtime FROM table where passtime<'2020-02-19 15:28:14' and passtime>'2020-02-18 15:28:37'
order by passtime desc limit 10;
SELECT max(passtime) FROM table where passtime<'2020-02-19 15:28:14' and passtime>'2020-02-18
15:28:37';
```

For a column-store table, the partition key is **createtime** and the hash distribution key is **motorvehicleid**.

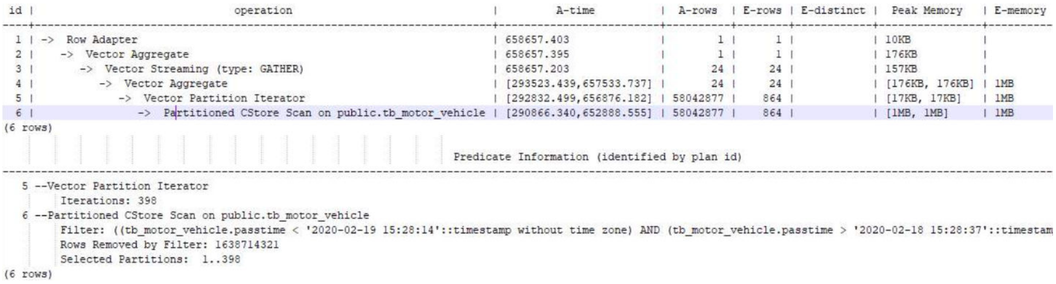
Possible Causes

The query criteria of slow SQL statements do not include the partition field. As a result, partitions are not pruned from the execution plan and the entire table is scanned, severely deteriorating performance.

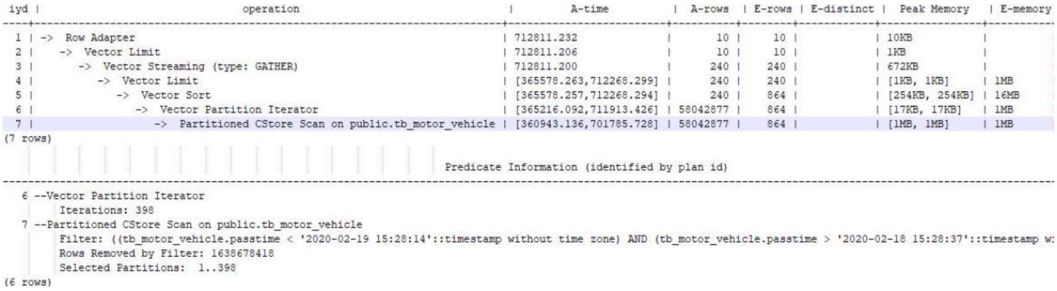
Cause Analysis

Step 1 Some services of the customer are slow, and all these services involve the same table **tb_motor_vehicle**.

Step 2 Collect several typical slow SQL statements and print their execution plans. The execution plans show that during the execution of the two SQL statements, time is mostly spent on the partition scanning of the **Partitioned CStore Scan on public.tb_motor_vehicle** column-store table.



```
 id |                               operation                               |           A-time           |  A-rows  | E-rows | E-distinct |  Peak Memory   | E-memory
----+-----------------------------------------------------------------------+----------------------------+----------+--------+------------+----------------+----------
  1 | ->  Row Adapter                                                       | 658657.403                 |        1 |      1 |            | 10KB           |
  2 |    ->  Vector Aggregate                                               | 658657.395                 |        1 |      1 |            | 176KB          |
  3 |       ->  Vector Streaming (type: GATHER)                             | 658657.203                 |       24 |     24 |            | 157KB          |
  4 |          ->  Vector Aggregate                                         | [293523.439,657533.737]    |       24 |     24 |            | [176KB, 176KB] | 1MB
  5 |             ->  Vector Partition Iterator                             | [292832.499,656876.182]    | 58042877 |    864 |            | [17KB, 17KB]   | 1MB
  6 |                ->  Partitioned CStore Scan on public.tb_motor_vehicle | [290866.340,652888.555]    | 58042877 |    864 |            | [1MB, 1MB]     | 1MB
(6 rows)

                                                    Predicate Information (identified by plan id)
------------------------------------------------------------------------------------------------------------------------------------------------------------------
   5 --Vector Partition Iterator
         Iterations: 398
   6 --Partitioned CStore Scan on public.tb_motor_vehicle
         Filter: ((tb_motor_vehicle.passtime < '2020-02-19 15:28:14'::timestamp without time zone) AND (tb_motor_vehicle.passtime > '2020-02-18 15:28:37'::timestam
         Rows Removed by Filter: 1638714321
         Selected Partitions:  1..398
(6 rows)
```

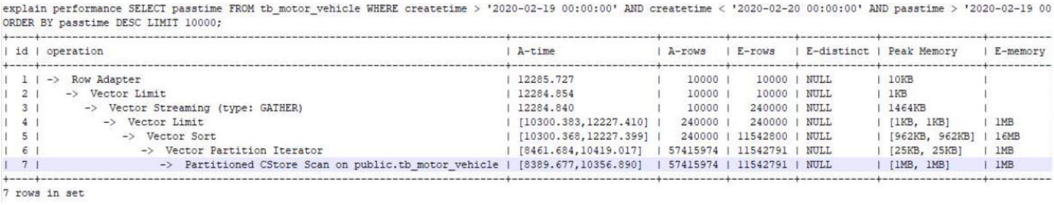


```
iyd |                                 operation                               |           A-time           |  A-rows  | E-rows | E-distinct |  Peak Memory   | E-memory
----+-------------------------------------------------------------------------+----------------------------+----------+--------+------------+----------------+----------
  1 | ->  Row Adapter                                                         | 712811.232                 |       10 |     10 |            | 10KB           |
  2 |    ->  Vector Limit                                                     | 712811.206                 |       10 |     10 |            | 1KB            |
  3 |       ->  Vector Streaming (type: GATHER)                               | 712811.200                 |      240 |    240 |            | 672KB          |
  4 |          ->  Vector Limit                                               | [365578.263,712268.299]    |      240 |    240 |            | [1KB, 1KB]     | 1MB
  5 |             ->  Vector Sort                                             | [365578.257,712268.294]    |      240 |    864 |            | [254KB, 254KB] | 16MB
  6 |                ->  Vector Partition Iterator                            | [365216.092,711913.426]    | 58042877 |    864 |            | [17KB, 17KB]   | 1MB
  7 |                   ->  Partitioned CStore Scan on public.tb_motor_vehicle | [360943.136,701785.728]    | 58042877 |    864 |            | [1MB, 1MB]     | 1MB
(7 rows)

                                                    Predicate Information (identified by plan id)
------------------------------------------------------------------------------------------------------------------------------------------------------------------
   6 --Vector Partition Iterator
         Iterations: 398
   7 --Partitioned CStore Scan on public.tb_motor_vehicle
         Filter: ((tb_motor_vehicle.passtime < '2020-02-19 15:28:14'::timestamp without time zone) AND (tb_motor_vehicle.passtime > '2020-02-18 15:28:37'::timestamp w:
         Rows Removed by Filter: 1638678418
         Selected Partitions:  1..398
(6 rows)
```

Step 3 According to the customer, the partition key of the table is **createtime**. However, the query criteria of the involved SQL statements do not contain **createtime**. It can be confirmed that partitions are not pruned from the execution plan of slow SQL statements. As a result, the entire table with 18.5 billion data records is scanned, and the scanning performance is poor.

Step 4 After the partition key is added to the query criteria, the optimized SQL statement and execution plan are as follows. The query duration is slashed from more than 10 minutes to about 12 seconds.

```
SELECT passtime FROM tb_motor_vehicle WHERE createtime > '2020-02-19 00:00:00' AND createtime <
'2020-02-20 00:00:00' AND passtime > '2020-02-19 00:00:00' AND passtime < '2020-02-20 00:00:00' ORDER
BY passtime DESC LIMIT 10000;
```



```
explain performance SELECT passtime FROM tb_motor_vehicle WHERE createtime > '2020-02-19 00:00:00' AND createtime < '2020-02-20 00:00:00' AND passtime > '2020-02-19 00
ORDER BY passtime DESC LIMIT 10000;
+----+---------------------------------------------------------------+------------------------+----------+----------+------------+----------------+---------
| id | operation                                                     | A-time                 | A-rows   | E-rows   | E-distinct | Peak Memory    | E-memory
+----+---------------------------------------------------------------+------------------------+----------+----------+------------+----------------+---------
|  1 | ->  Row Adapter                                               | 12285.727              |    10000 |    10000 | NULL       | 10KB           |
|  2 |   ->  Vector Limit                                            | 12284.854              |    10000 |    10000 | NULL       | 1KB            |
|  3 |     ->  Vector Streaming (type: GATHER)                       | 12284.840              |    10000 |   240000 | NULL       | 1464KB         |
|  4 |       ->  Vector Limit                                        | [10300.383,12227.410]  |   240000 |   240000 | NULL       | [1KB, 1KB]     | 1MB
|  5 |         ->  Vector Sort                                       | [10300.368,12227.399]  |   240000 | 11542800 | NULL       | [962KB, 962KB] | 16MB
|  6 |           ->  Vector Partition Iterator                       | [8461.684,10419.017]   | 57415974 | 11542791 | NULL       | [25KB, 25KB]   | 1MB
|  7 |             ->  Partitioned CStore Scan on public.tb_motor_vehicle | [8389.677,10356.890] | 57415974 | 11542791 | NULL       | [1MB, 1MB]     | 1MB
+----+---------------------------------------------------------------+------------------------+----------+----------+------------+----------------+---------
7 rows in set
```

----End

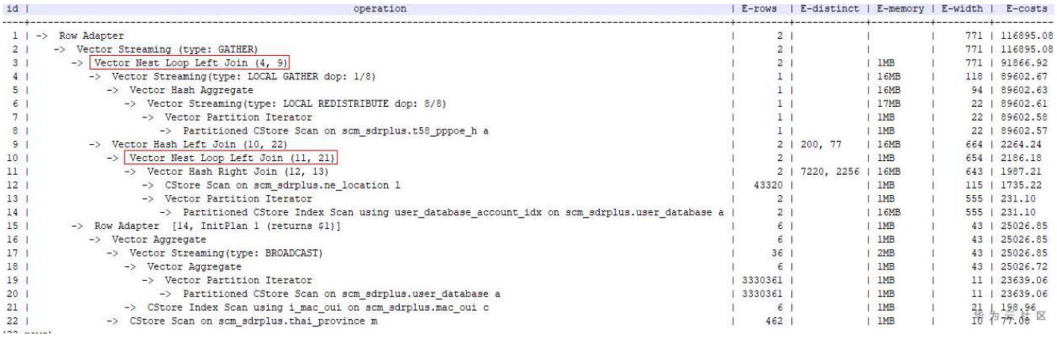
Handling Procedure

Add the partition query criteria to the slow SQL statements to prevent full table scanning.

6.21 Optimizer Uses Nested Loop Due to the Small Estimated Number of Rows and the Performance Deteriorates

Symptom

The query statement execution is slow and the query result cannot be returned. For SQL statements, the **LEFT JOIN** statement is used to query data from two or three tables and then the **SELECT** statement is used to query the result. The execution plan is as follows.



```
id |                                  operation                                                       | E-rows  | E-distinct | E-memory | E-width |  E-costs
---+--------------------------------------------------------------------------------------------------+---------+------------+----------+---------+-----------
 1 | ->  Row Adapter                                                                                  |       2 |            |          |     771 | 116895.08
 2 |   ->  Vector Streaming (type: GATHER)                                                            |       2 |            |          |     771 | 116895.08
 3 |     ->  Vector Nest Loop Left Join (4, 9)                                                        |       2 |            | 1MB      |     771 | 91866.92
 4 |       ->  Vector Streaming(type: LOCAL GATHER dop: 1/8)                                          |       1 |            | 16MB     |     118 | 89602.67
 5 |         ->  Vector Hash Aggregate                                                                |       1 |            | 16MB     |      94 | 89602.63
 6 |           ->  Vector Streaming(type: LOCAL REDISTRIBUTE dop: 8/8)                                |       1 |            | 17MB     |      22 | 89602.61
 7 |             ->  Vector Partition Iterator                                                        |       1 |            | 1MB      |      22 | 89602.58
 8 |               ->  Partitioned CStore Scan on scm_sdrplus.t58_pppoe_h a                           |       1 |            | 1MB      |      22 | 89602.57
 9 |       ->  Vector Hash Left Join (10, 22)                                                         |       2 | 200, 77    | 16MB     |     664 | 2264.24
10 |         ->  Vector Nest Loop Left Join (11, 21)                                                  |       2 |            | 1MB      |     654 | 2186.18
11 |           ->  Vector Hash Right Join (12, 13)                                                    |       2 | 7220, 2256 | 16MB     |     643 | 1987.21
12 |             ->  CStore Scan on scm_sdrplus.ne_location l                                         |   43320 |            | 1MB      |     115 | 1735.22
13 |             ->  Vector Partition Iterator                                                        |       2 |            | 1MB      |     555 | 231.10
14 |               ->  Partitioned CStore Index Scan using user_database_account_idx on scm_sdrplus.user_database a |       2 |            | 16MB     |     555 | 231.10
15 |     ->  Row Adapter  [14, InitPlan 1 (returns $1)]                                               |       6 |            | 1MB      |      43 | 25026.85
16 |       ->  Vector Aggregate                                                                       |       6 |            | 1MB      |      43 | 25026.85
17 |         ->  Vector Streaming(type: BROADCAST)                                                    |      36 |            | 2MB      |      43 | 25026.85
18 |           ->  Vector Aggregate                                                                   |       6 |            | 1MB      |      43 | 25026.72
19 |             ->  Vector Partition Iterator                                                        | 3330361 |            | 1MB      |      11 | 23639.06
20 |               ->  Partitioned CStore Scan on scm_sdrplus.user_database a                         | 3330361 |            | 1MB      |      11 | 23639.06
21 |           ->  CStore Index Scan using i_mac_oui on scm_sdrplus.mac_oui c                         |       6 |            | 1MB      |      21 | 198.96
22 |         ->  CStore Scan on scm_sdrplus.thai_province m                                           |     462 |            | 1MB      |      10 | 77.08
```

Possible Causes

When the optimizer selects an execution plan, the estimated number of result sets is small. As a result, nested loop is used and the performance deteriorates.

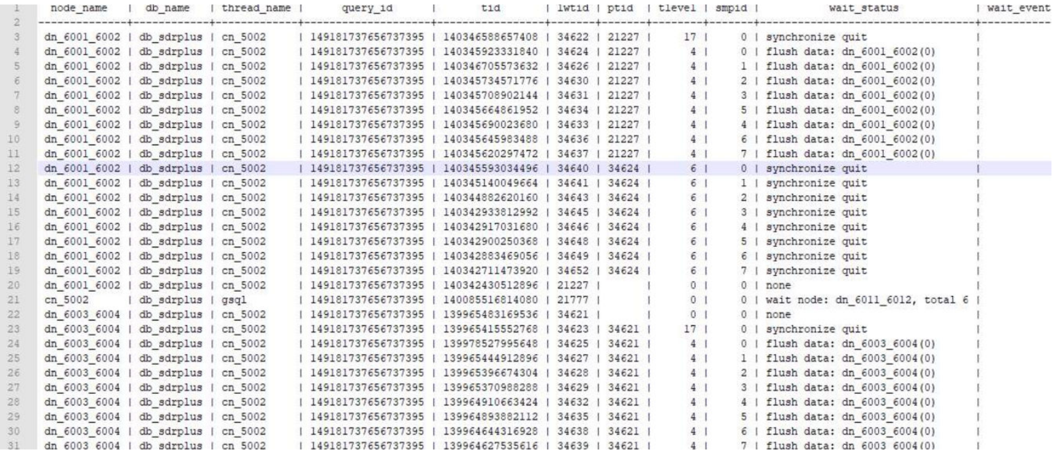
Cause Analysis

Step 1 Check the I/O, memory, and CPU usage. The resource usage of these indicators is not high.

Step 2 Check the thread waiting status of the slow SQL statements.

According to the thread waiting status, not all threads are waiting for processing on the same DN. Therefore, the intermediate result sets are not skewed on the same DN.

```
SELECT * FROM pg_thread_wait_status WHERE query_id='149181737656737395';
```

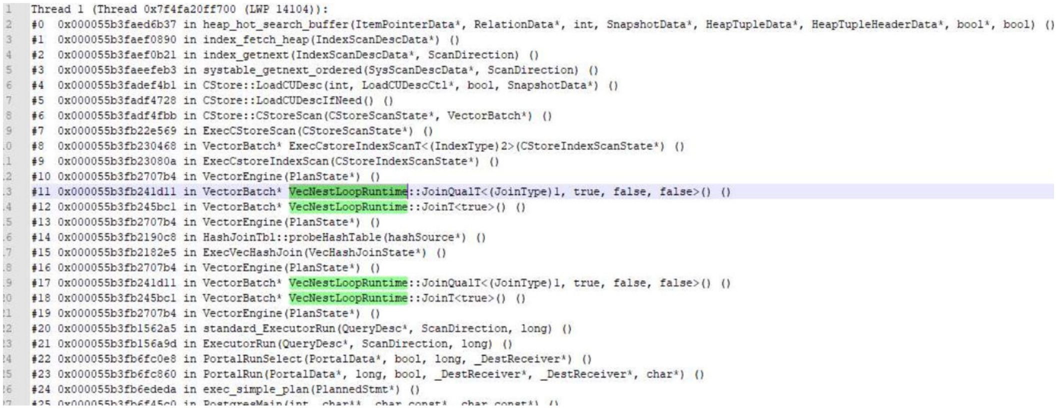


```
  node_name   |  db_name   | thread_name |      query_id      |       tid       | lwtid | ptid  | tlevel | smpid |           wait_status            | wait_event
--------------+------------+-------------+--------------------+-----------------+-------+-------+--------+-------+----------------------------------+-----------
 dn_6001_6002 | db_sdrplus | cn_5002     | 149181737656737395 | 140346588657408 | 34622 | 21227 |     17 |     0 | synchronize quit                 |
 dn_6001_6002 | db_sdrplus | cn_5002     | 149181737656737395 | 140345923331840 | 34624 | 21227 |      4 |     0 | flush data: dn_6001_6002(0)      |
 dn_6001_6002 | db_sdrplus | cn_5002     | 149181737656737395 | 140346705573632 | 34626 | 21227 |      4 |     1 | flush data: dn_6001_6002(0)      |
 dn_6001_6002 | db_sdrplus | cn_5002     | 149181737656737395 | 140345734571776 | 34630 | 21227 |      4 |     2 | flush data: dn_6001_6002(0)      |
 dn_6001_6002 | db_sdrplus | cn_5002     | 149181737656737395 | 140345708902144 | 34631 | 21227 |      4 |     3 | flush data: dn_6001_6002(0)      |
 dn_6001_6002 | db_sdrplus | cn_5002     | 149181737656737395 | 140345664861952 | 34634 | 21227 |      4 |     5 | flush data: dn_6001_6002(0)      |
 dn_6001_6002 | db_sdrplus | cn_5002     | 149181737656737395 | 140345690023680 | 34633 | 21227 |      4 |     4 | flush data: dn_6001_6002(0)      |
 dn_6001_6002 | db_sdrplus | cn_5002     | 149181737656737395 | 140345645983488 | 34636 | 21227 |      4 |     6 | flush data: dn_6001_6002(0)      |
 dn_6001_6002 | db_sdrplus | cn_5002     | 149181737656737395 | 140345620297472 | 34637 | 21227 |      4 |     7 | flush data: dn_6001_6002(0)      |
 dn_6001_6002 | db_sdrplus | cn_5002     | 149181737656737395 | 140345593034496 | 34640 | 34624 |      6 |     0 | synchronize quit                 |
 dn_6001_6002 | db_sdrplus | cn_5002     | 149181737656737395 | 140345140049664 | 34641 | 34624 |      6 |     1 | synchronize quit                 |
 dn_6001_6002 | db_sdrplus | cn_5002     | 149181737656737395 | 140344882620160 | 34643 | 34624 |      6 |     2 | synchronize quit                 |
 dn_6001_6002 | db_sdrplus | cn_5002     | 149181737656737395 | 140342933812992 | 34645 | 34624 |      6 |     3 | synchronize quit                 |
 dn_6001_6002 | db_sdrplus | cn_5002     | 149181737656737395 | 140342917031680 | 34646 | 34624 |      6 |     4 | synchronize quit                 |
 dn_6001_6002 | db_sdrplus | cn_5002     | 149181737656737395 | 140342900250368 | 34648 | 34624 |      6 |     5 | synchronize quit                 |
 dn_6001_6002 | db_sdrplus | cn_5002     | 149181737656737395 | 140342883469056 | 34649 | 34624 |      6 |     6 | synchronize quit                 |
 dn_6001_6002 | db_sdrplus | cn_5002     | 149181737656737395 | 140342711473920 | 34652 | 34624 |      6 |     7 | synchronize quit                 |
 dn_6001_6002 | db_sdrplus | cn_5002     | 149181737656737395 | 140342430512896 | 21227 |       |      0 |     0 | none                             |
 cn_5002      | db_sdrplus | gsql        | 149181737656737395 | 140085516814080 | 21777 |       |      0 |     0 | wait node: dn_6011_6012, total 6 |
 dn_6003_6004 | db_sdrplus | cn_5002     | 149181737656737395 | 139965483169536 | 34621 |       |      0 |     0 | none                             |
 dn_6003_6004 | db_sdrplus | cn_5002     | 149181737656737395 | 139965415552768 | 34623 | 34621 |     17 |     0 | synchronize quit                 |
 dn_6003_6004 | db_sdrplus | cn_5002     | 149181737656737395 | 139978527995648 | 34625 | 34621 |      4 |     0 | flush data: dn_6003_6004(0)      |
 dn_6003_6004 | db_sdrplus | cn_5002     | 149181737656737395 | 139965444912896 | 34627 | 34621 |      4 |     1 | flush data: dn_6003_6004(0)      |
 dn_6003_6004 | db_sdrplus | cn_5002     | 149181737656737395 | 139965396674304 | 34628 | 34621 |      4 |     2 | flush data: dn_6003_6004(0)      |
 dn_6003_6004 | db_sdrplus | cn_5002     | 149181737656737395 | 139965370988288 | 34629 | 34621 |      4 |     3 | flush data: dn_6003_6004(0)      |
 dn_6003_6004 | db_sdrplus | cn_5002     | 149181737656737395 | 139964910663424 | 34632 | 34621 |      4 |     4 | flush data: dn_6003_6004(0)      |
 dn_6003_6004 | db_sdrplus | cn_5002     | 149181737656737395 | 139964893882112 | 34635 | 34621 |      4 |     5 | flush data: dn_6003_6004(0)      |
 dn_6003_6004 | db_sdrplus | cn_5002     | 149181737656737395 | 139964644316928 | 34638 | 34621 |      4 |     6 | flush data: dn_6003_6004(0)      |
 dn_6003_6004 | db_sdrplus | cn_5002     | 149181737656737395 | 139964627535616 | 34639 | 34621 |      4 |     7 | flush data: dn_6003_6004(0)      |
```

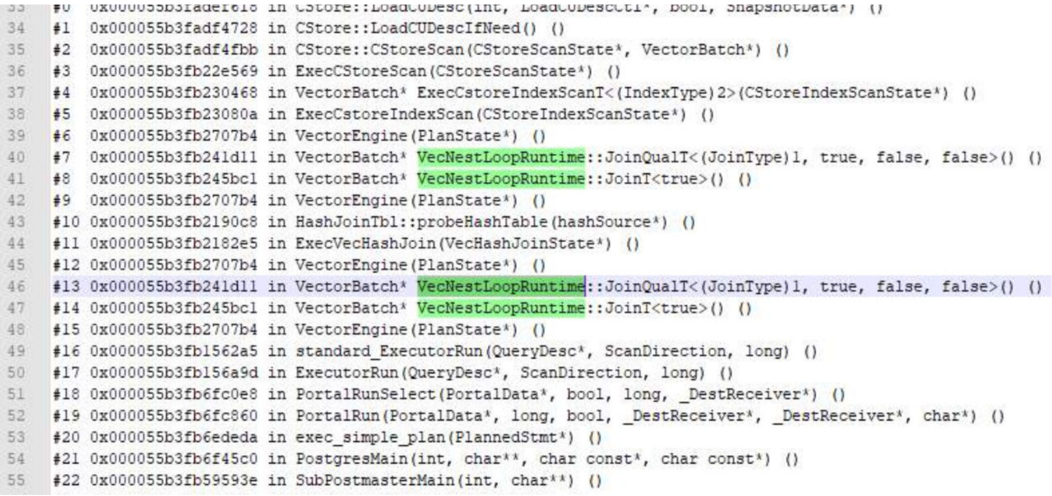
Step 3 Contact O&M personnel to log in to the corresponding instance node and print the stack information about the threads whose waiting status is **none**.

After the stack information is repeatedly printed, it is found that the stack changes and does not hang. The problem may be caused by slow performance. In addition, **VecNestLoopRuntime** exists in the stack. It is determined that the performance deteriorates because the execution plan uses nested loop. This occurs because the statistics are inaccurate and the number of result sets estimated by the optimizer is small.

gstack *14104*



```
Thread 1 (Thread 0x7f4fa20ff700 (LWP 14104)):
#0  0x000055b3faed6b37 in heap_hot_search_buffer(ItemPointerData*, RelationData*, int, SnapshotData*, HeapTupleData*, HeapTupleHeaderData*, bool*, bool) ()
#1  0x000055b3faef0890 in index_fetch_heap(IndexScanDescData*) ()
#2  0x000055b3faef0b21 in index_getnext(IndexScanDescData*, ScanDirection) ()
#3  0x000055b3faeefeb3 in systable_getnext_ordered(SysScanDescData*, ScanDirection) ()
#4  0x000055b3fadef4b1 in CStore::LoadCUDesc(int, LoadCUDescCtl*, bool, SnapshotData*) ()
#5  0x000055b3fadf4728 in CStore::LoadCUDescIfNeed() ()
#6  0x000055b3fadf4fbb in CStore::CStoreScan(CStoreScanState*, VectorBatch*) ()
#7  0x000055b3fb22e569 in ExecCStoreScan(CStoreScanState*) ()
#8  0x000055b3fb230468 in VectorBatch* ExecCstoreIndexScanT<(IndexType)2>(CStoreIndexScanState*) ()
#9  0x000055b3fb23080a in ExecCstoreIndexScan(CStoreIndexScanState*) ()
#10 0x000055b3fb2707b4 in VectorEngine(PlanState*) ()
#11 0x000055b3fb241d11 in VectorBatch* VecNestLoopRuntime::JoinQualT<(JoinType)1, true, false, false>() ()
#12 0x000055b3fb245bc1 in VectorBatch* VecNestLoopRuntime::JoinT<true>() ()
#13 0x000055b3fb2707b4 in VectorEngine(PlanState*) ()
#14 0x000055b3fb2190c8 in HashJoinTbl::probeHashTable(hashSource*) ()
#15 0x000055b3fb2182e5 in ExecVecHashJoin(VecHashJoinState*) ()
#16 0x000055b3fb2707b4 in VectorEngine(PlanState*) ()
#17 0x000055b3fb241d11 in VectorBatch* VecNestLoopRuntime::JoinQualT<(JoinType)1, true, false, false>() ()
#18 0x000055b3fb245bc1 in VectorBatch* VecNestLoopRuntime::JoinT<true>() ()
#19 0x000055b3fb2707b4 in VectorEngine(PlanState*) ()
#20 0x000055b3fb1562a5 in standard_ExecutorRun(QueryDesc*, ScanDirection, long) ()
#21 0x000055b3fb156a9d in ExecutorRun(QueryDesc*, ScanDirection, long) ()
#22 0x000055b3fb6fc0e8 in PortalRunSelect(PortalData*, bool, long, _DestReceiver*) ()
#23 0x000055b3fb6fc860 in PortalRun(PortalData*, long, bool, _DestReceiver*, _DestReceiver*, char*) ()
#24 0x000055b3fb6ededa in exec_simple_plan(PlannedStmt*) ()
#25 0x000055b3fb6f45c0 in PostgresMain(int, char**, char const*, char const*) ()
```

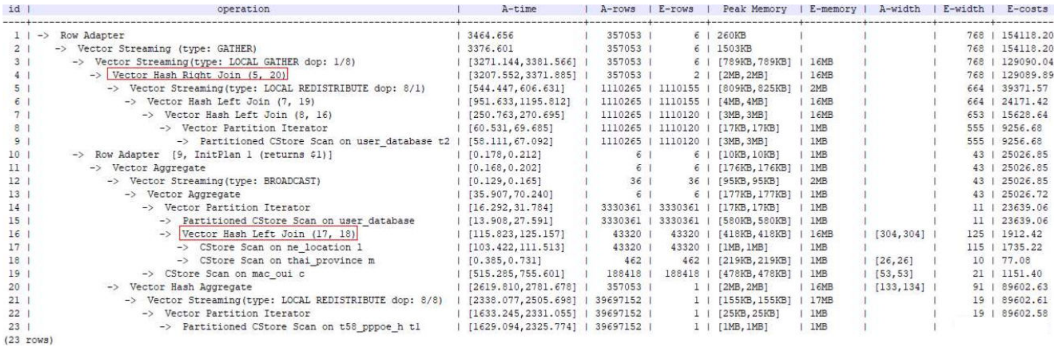


```
#0  0x000055b3fadef618 in CStore::LoadCUDesc(int, LoadCUDescCtl*, bool, SnapshotData*) ()
#1  0x000055b3fadf4728 in CStore::LoadCUDescIfNeed() ()
#2  0x000055b3fadf4fbb in CStore::CStoreScan(CStoreScanState*, VectorBatch*) ()
#3  0x000055b3fb22e569 in ExecCStoreScan(CStoreScanState*) ()
#4  0x000055b3fb230468 in VectorBatch* ExecCstoreIndexScanT<(IndexType)2>(CStoreIndexScanState*) ()
#5  0x000055b3fb23080a in ExecCstoreIndexScan(CStoreIndexScanState*) ()
#6  0x000055b3fb2707b4 in VectorEngine(PlanState*) ()
#7  0x000055b3fb241d11 in VectorBatch* VecNestLoopRuntime::JoinQualT<(JoinType)1, true, false, false>() ()
#8  0x000055b3fb245bc1 in VectorBatch* VecNestLoopRuntime::JoinT<true>() ()
#9  0x000055b3fb2707b4 in VectorEngine(PlanState*) ()
#10 0x000055b3fb2190c8 in HashJoinTbl::probeHashTable(hashSource*) ()
#11 0x000055b3fb2182e5 in ExecVecHashJoin(VecHashJoinState*) ()
#12 0x000055b3fb2707b4 in VectorEngine(PlanState*) ()
#13 0x000055b3fb241d11 in VectorBatch* VecNestLoopRuntime::JoinQualT<(JoinType)1, true, false, false>() ()
#14 0x000055b3fb245bc1 in VectorBatch* VecNestLoopRuntime::JoinT<true>() ()
#15 0x000055b3fb2707b4 in VectorEngine(PlanState*) ()
#16 0x000055b3fb1562a5 in standard_ExecutorRun(QueryDesc*, ScanDirection, long) ()
#17 0x000055b3fb156a9d in ExecutorRun(QueryDesc*, ScanDirection, long) ()
#18 0x000055b3fb6fc0e8 in PortalRunSelect(PortalData*, bool, long, _DestReceiver*) ()
#19 0x000055b3fb6fc860 in PortalRun(PortalData*, long, bool, _DestReceiver*, _DestReceiver*, char*) ()
#20 0x000055b3fb6ededa in exec_simple_plan(PlannedStmt*) ()
#21 0x000055b3fb6f45c0 in PostgresMain(int, char**, char const*, char const*) ()
#22 0x000055b3fb59593e in SubPostmasterMain(int, char**) ()
```

Step 4 The performance is not improved after **ANALYZE** is executed on the table.

Step 5 After hints are added to SQL statements to disable the index function and the optimizer forcibly executes hash join, the hint function does not take effect because the hints cannot change the plan in the subquery.

Step 6 After **SET enable_indexscan** is set to **off**, the execution plan is changed and **HASH LEFT JOIN** is used. The execution result of the slow SQL statement is displayed in about 3 seconds, meeting the customer's requirements.



```
 id |                              operation                               |        A-time         |  A-rows  | E-rows  |  Peak Memory   | E-memory |  A-width   | E-width |  E-costs
----+----------------------------------------------------------------------+-----------------------+----------+---------+----------------+----------+------------+---------+-----------
  1 | ->  Row Adapter                                                      | 3464.656              |   357053 |       6 | 260KB          |          |            |     768 | 154118.20
  2 |    ->  Vector Streaming (type: GATHER)                               | 3376.601              |   357053 |       6 | 1503KB         |          |            |     768 | 154118.20
  3 |       ->  Vector Streaming(type: LOCAL GATHER dop: 1/8)              | [3271.144,3381.566]   |   357053 |       6 | [789KB,789KB]  | 16MB     |            |     768 | 129090.04
  4 |          ->  Vector Hash Right Join (5, 20)                          | [3207.552,3371.885]   |   357053 |       2 | [2MB,2MB]      | 16MB     |            |     768 | 129089.89
  5 |             ->  Vector Streaming(type: LOCAL REDISTRIBUTE dop: 8/1)  | [544.447,606.631]     |  1110265 | 1110155 | [809KB,825KB]  | 2MB      |            |     664 | 39371.57
  6 |                ->  Vector Hash Left Join (7, 19)                     | [951.633,1195.812]    |  1110265 | 1110155 | [4MB,4MB]      | 16MB     |            |     664 | 24171.42
  7 |                   ->  Vector Hash Left Join (8, 16)                  | [250.763,270.695]     |  1110265 | 1110120 | [3MB,3MB]      | 16MB     |            |     653 | 15628.64
  8 |                      ->  Vector Partition Iterator                   | [60.531,69.685]       |  1110265 | 1110120 | [17KB,17KB]    | 1MB      |            |     555 | 9256.68
  9 |                         ->  Partitioned CStore Scan on user_database t2 | [58.111,67.092]    |  1110265 | 1110120 | [3MB,3MB]      | 1MB      |            |     555 | 9256.68
 10 |       ->  Row Adapter  [9, InitPlan 1 (returns $1)]                  | [0.178,0.212]         |        6 |       6 | [10KB,10KB]    | 1MB      |            |      43 | 25026.85
 11 |          ->  Vector Aggregate                                        | [0.168,0.202]         |        6 |       6 | [176KB,176KB]  | 1MB      |            |      43 | 25026.85
 12 |             ->  Vector Streaming(type: BROADCAST)                    | [0.129,0.165]         |       36 |      36 | [95KB,95KB]    | 2MB      |            |      43 | 25026.85
 13 |                ->  Vector Aggregate                                  | [35.907,70.240]       |        6 |       6 | [177KB,177KB]  | 1MB      |            |      43 | 25026.72
 14 |                   ->  Vector Partition Iterator                      | [16.292,31.784]       |  3330361 | 3330361 | [17KB,17KB]    | 1MB      |            |      11 | 23639.06
 15 |                      ->  Partitioned CStore Scan on user_database    | [13.908,27.591]       |  3330361 | 3330361 | [580KB,580KB]  | 1MB      |            |      11 | 23639.06
 16 |                      -> Vector Hash Left Join (17, 18)               | [115.823,125.157]     |    43320 |   43320 | [418KB,418KB]  | 16MB     | [304,304]  |     125 | 1912.42
 17 |                         ->  CStore Scan on ne_location l             | [103.422,111.513]     |    43320 |   43320 | [1MB,1MB]      | 1MB      |            |     115 | 1735.22
 18 |                         ->  CStore Scan on thai_province m           | [0.385,0.731]         |      462 |     462 | [219KB,219KB]  | 1MB      | [26,26]    |      10 | 77.08
 19 |                   ->  CStore Scan on mac_oui c                       | [515.285,755.601]     |   188418 |  188418 | [478KB,478KB]  | 1MB      | [53,53]    |      21 | 1151.40
 20 |             ->  Vector Hash Aggregate                                | [2619.810,2781.678]   |   357053 |       1 | [2MB,2MB]      | 16MB     | [133,134]  |      91 | 89602.63
 21 |                ->  Vector Streaming(type: LOCAL REDISTRIBUTE dop: 8/8) | [2338.077,2505.698] | 39697152 |       1 | [155KB,155KB]  | 17MB     |            |      19 | 89602.61
 22 |                   ->  Vector Partition Iterator                      | [1633.245,2331.055]   | 39697152 |       1 | [25KB,25KB]    | 1MB      |            |      19 | 89602.58
 23 |                      ->  Partitioned CStore Scan on t58_pppoe_h tl   | [1629.094,2325.774]   | 39697152 |       1 | [1MB,1MB]      | 1MB      |            |
(23 rows)
```

----End

Handling Procedure

Set **enable_indexscan** to **off** to disable the index function so that the execution plan generated by the optimizer uses hash join instead of nested loop.

6.22 SQL Statements Contain the in Constant and No Result Is Returned After SQL Statement Execution

Symptom

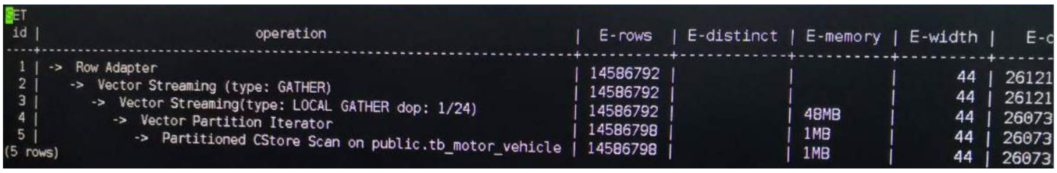
The **in** constant is one of the criteria of the SQL statement for filtering large tables. There are more than 2000 constants. The base table contains a large amount of data. No result is returned after the SQL statement is executed.

Possible Causes

The **in** condition still exists as a common filtering condition in the execution plan. In this scenario, the performance of the **join** operation is better than that of the **in** constant. You need to use the **join** operation instead of the **in** constant for better performance.

Cause Analysis

Step 1 Print the statement execution plan.



```
SET
 id |                    operation                        |  E-rows  | E-distinct | E-memory | E-width |   E-c
----+-----------------------------------------------------+----------+------------+----------+---------+------
  1 | -> Row Adapter                                      | 14586792 |            |          |      44 | 26121
  2 |   -> Vector Streaming (type: GATHER)                | 14586792 |            |          |      44 | 26121
  3 |     -> Vector Streaming(type: LOCAL GATHER dop: 1/24) | 14586792 |          | 48MB     |      44 | 26073
  4 |       -> Vector Partition Iterator                  | 14586798 |            | 1MB      |      44 | 26073
  5 |         -> Partitioned CStore Scan on public.tb_motor_vehicle | 14586798 | | 1MB   |      44 | 26073
(5 rows)
```

Step 2 The **in** condition still exists as a common filtering condition in the execution plan. In this scenario, the performance of the **join** operation is better than that of the **in** constant. You need to use the **join** operation instead of the **in** constant for better performance.

----End

Handling Procedure

Step 1 The default value of **qrw_inlist2join_optmode** is **cost_base**. You can change the **in** constant to a join operation. If the number of rows estimated by the optimizer is inaccurate, the value of the parameter may not be changed in some scenarios, resulting in poor performance.

Step 2 To solve this problem, set **qrw_inlist2join_optmode** to **rule_base**.

```
set qrw_inlist2join_optmode to rule_base;
```

----End

6.23 Performance of Single-Table Point Query Is Poor

Symptom

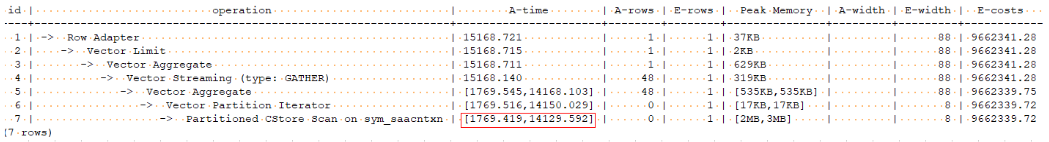
The customer expects the result of single-table query to be returned within 1 second. However, the execution takes more than 10 seconds.

Possible Causes

This problem occurs because incorrect row- and column-store tables are selected. In this scenario, the row-store table and B-tree index should be used.

Cause Analysis

Step 1 The execution information about the faulty SQL statements shows that most of the time is spent on CStore Scan.



```
 id |                          operation                           |        A-time         | A-rows | E-rows |  Peak Memory   | A-width | E-width |  E-costs
----+--------------------------------------------------------------+-----------------------+--------+--------+----------------+---------+---------+------------
  1 | ->  Row Adapter                                              | 15168.721             |      1 |      1 | 37KB           |         |      88 | 9662341.28
  2 |    ->  Vector Limit                                          | 15168.715             |      1 |      1 | 2KB            |         |      88 | 9662341.28
  3 |       ->  Vector Aggregate                                   | 15168.711             |      1 |      1 | 629KB          |         |      88 | 9662341.28
  4 |          ->  Vector Streaming (type: GATHER)                 | 15168.140             |     48 |      1 | 319KB          |         |      88 | 9662341.28
  5 |             ->  Vector Aggregate                             | [1769.545,14168.103]  |     48 |      1 | [535KB,535KB]  |         |      88 | 9662339.75
  6 |                ->  Vector Partition Iterator                 | [1769.516,14150.029]  |      0 |      1 | [17KB,17KB]    |         |       8 | 9662339.72
  7 |                   ->  Partitioned CStore Scan on sym_saacntxn | [1769.419,14129.592] |      0 |      1 | [2MB,3MB]      |         |       8 | 9662339.72
(7 rows)
```

Step 2 The base table contains more than one billion data records. Incremental data is imported to the database in batches every night, and a small amount of data is cleaned. In the daytime, a large number of concurrent query operations are performed. The query does not involve table association, and the values of returned results are not large.

----End

Handling Procedure

Step 1 Adjust the table definition, change the table to a row-store table, and create a B-tree index. The principles for creating a B-tree index are as follows.

1. Analyze the customer's SQL statements before creating a B-tree index.
2. Do not create redundant indexes.
3. Place the columns with better filtering performance in the front of the index.
4. Include as many filter criteria as possible in the index.

----End

6.24 CCN Queuing Under Dynamic Load Management

Symptom

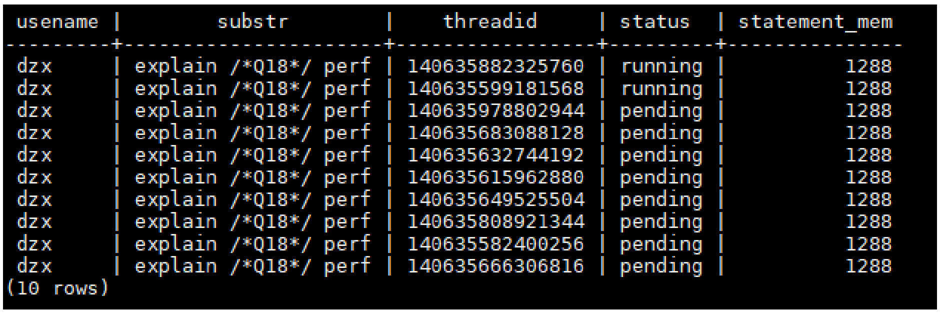
Services are running slowly. Only a few statements are being executed, and other service statements are waiting in the CCN queue.

Possible Causes

In dynamic load management, statements are sorted based on the estimated memory. For example, if the maximum available dynamic memory is 10 GB (per instance) and the estimated memory used by a statement is 5 GB, a maximum of two statements can be executed at the same time, and other statements have to wait in the CCN queue.

Solution

- Scenario 1: The estimated statement memory is too large. Statements are queuing.
  – Query the **pg_session_wlmstat** view to check whether there are only a few statements in the running state, and whether the value of **statement_mem** is large. (The unit is MB. Generally, statements whose estimated memory usage is greater than 1/3 of **max_dynamic_memory** are large-memory statements.) If all these conditions are met, the slow execution is caused by the statements that occupy too much memory.
    ```
    SELECT usename,substr(query,0,20),threadid,status,statement_mem FROM pg_session_wlmstat
    where usename not in ('omm','Ruby') order by statement_mem,status desc;
    ```



```
 usename |        substr         |     threadid     | status  | statement_mem
---------+-----------------------+------------------+---------+---------------
 dzx     | explain /*Q18*/ perf | 140635882325760 | running |          1288
 dzx     | explain /*Q18*/ perf | 140635599181568 | running |          1288
 dzx     | explain /*Q18*/ perf | 140635978802944 | pending |          1288
 dzx     | explain /*Q18*/ perf | 140635683088128 | pending |          1288
 dzx     | explain /*Q18*/ perf | 140635632744192 | pending |          1288
 dzx     | explain /*Q18*/ perf | 140635615962880 | pending |          1288
 dzx     | explain /*Q18*/ perf | 140635649525504 | pending |          1288
 dzx     | explain /*Q18*/ perf | 140635808921344 | pending |          1288
 dzx     | explain /*Q18*/ perf | 140635582400256 | pending |          1288
 dzx     | explain /*Q18*/ perf | 140635666306816 | pending |          1288
(10 rows)
```

As shown in the preceding figure, only the last statement is in the running state, and other statements are in the pending state. The **statement_mem** column shows that the running statement occupies 2576 MB memory. In this case, run the following statement to kill the thread based on the thread ID of the statement. After the thread is killed, resources will be released for other statements to run.

```
SELECT pg_terminate_backend(threadid);
```

- Scenario 2: All the statements are in the pending state. No statements are running. This is because the management and control mechanism is abnormal. You can kill all the threads to rectify the fault.

  ```
  SELECT pg_terminate_backend(pid) FROM pg_stat_activity where usename not in ('omm','Ruby');
  ```

6.25 Performance Deterioration Due to Data Bloat

Symptom

Data bloat causes disk space to be insufficient, thus deteriorating performance.

Possible Causes

You can run the **VACUUM**/**VACUUM FULL** command on the management plane to periodically reclaim space.

- Frequent table creation and deletion can lead to table bloating. To free up space, you can run the **VACUUM** command on system catalogs.
- Frequently update and delete operations can lead to table bloating. To free up space, you can run the **VACUUM** or **VACUUM FULL** command on system catalogs.

  NOTE

  Only 8.1.3 and later cluster versions support this function.

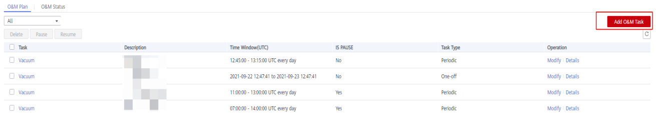
Handling Procedure

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

Step 2 Click the name of the target cluster.

Step 3 In the navigation pane, choose **Intelligent O&M**.

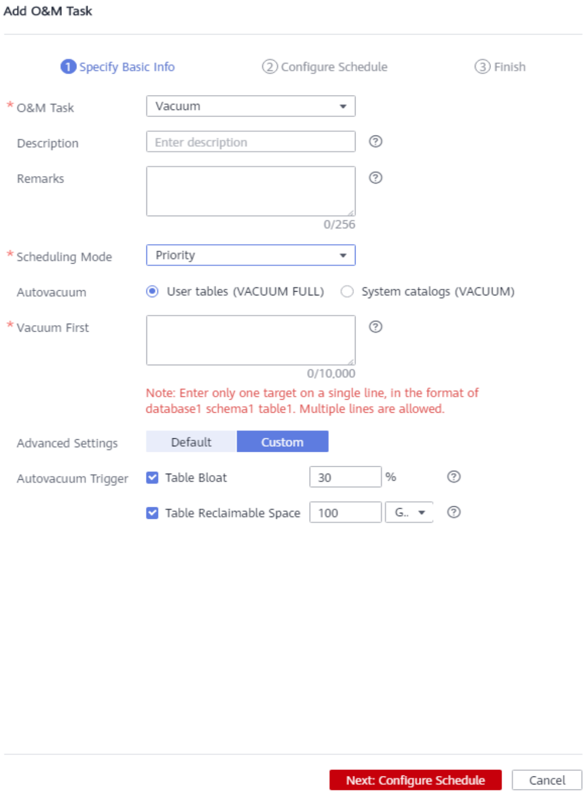
Step 4 Click the **O&M Plan** tab. Click **Add O&M Task**.



☐ Task	Description	Time Window(UTC)	IS PAUSE	Task Type	Operation
☐ Vacuum	[redacted]	12:45:00 - 13:15:00 UTC every day	No	Periodic	Modify | Details
☐ Vacuum	[redacted]	2021-09-22 12:47:41 to 2021-09-23 12:47:41	No	One-off	Modify | Details
☐ Vacuum	[redacted]	11:00:00 - 13:00:00 UTC every day	Yes	Periodic	Modify | Details
☐ Vacuum	[redacted]	07:00:00 - 14:00:00 UTC every day	Yes	Periodic	Modify | Details

Step 5 The **Add O&M Task** page is displayed.

- Select **Vacuum** for **O&M Task**.
- Set **Scheduling Mode** to **Auto**. GaussDB(DWS) automatically scans tables that require **VACUUM** operation.
- Select **System catalogs** or **User tables** for **Autovacuum**.
  - If there are a large number of **UPDATE** and **DELETE** operations, select the **User tables**.
  - If there are a large number of **CREATE** and **DELETE** operations, select **System catalogs**.



Add O&M Task

* O&M Task: Vacuum

Description: Enter description

Remarks: 0/256

* Scheduling Mode: Priority

Autovacuum: ● User tables (VACUUM FULL) ○ System catalogs (VACUUM)

* Vacuum First: 0/10,000

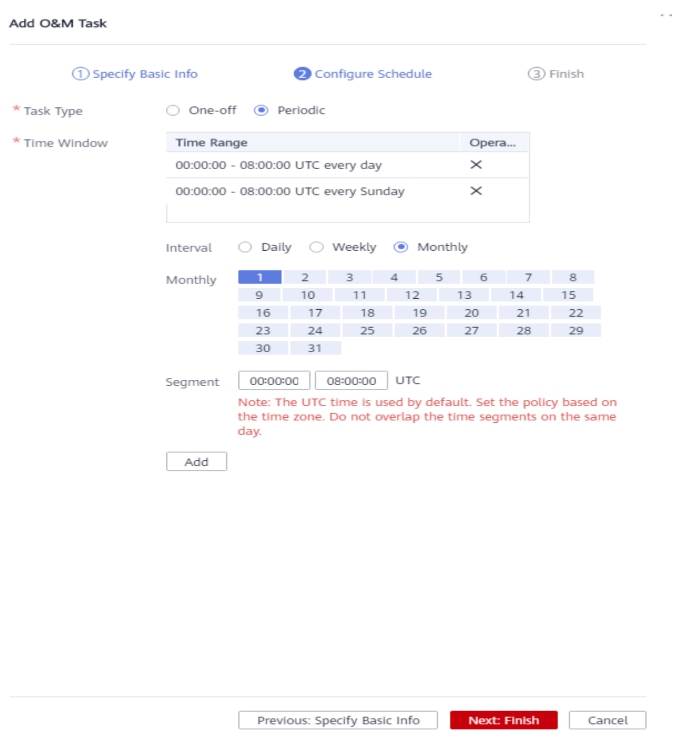
Note: Enter only one target on a single line, in the format of database1 schema1 table1. Multiple lines are allowed.

Advanced Settings: Default | Custom

Autovacuum Trigger: ☑ Table Bloat 30 %

☑ Table Reclaimable Space 100 G..

Step 6 Click **Next: Configure Schedule** to configure the schedule and Vacuum type. You are advised to select **Periodic** for **Task Type**. The GaussDB(DWS) automatically executes VACUUM in your selected time windows.



Add O&M Task

* Task Type ○ One-off ● Periodic

* Time Window

Time Range	Opera...
00:00:00 - 08:00:00 UTC every day	✕
00:00:00 - 08:00:00 UTC every Sunday	✕

Interval ○ Daily ○ Weekly ● Monthly

Monthly

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	
16	17	18	19	20	21	22	
23	24	25	26	27	28	29	
30	31						

Segment 00:00:00 08:00:00 UTC

Note: The UTC time is used by default. Set the policy based on the time zone. Do not overlap the time segments on the same day.

NOTE

For automatic Vacuum O&M tasks, the system uses the **VACUUM FULL** operation to process user tables. VACUUM FULL holds a level 8 lock, which blocks other transactions. Other transactions will be in lock waiting during **VACUUM FULL** execution. After 20 minutes, a timeout error is reported. Therefore, do not perform other transactions in the configured time window.

Step 7 After confirming that the information is correct, click **Next** to complete the configuration.

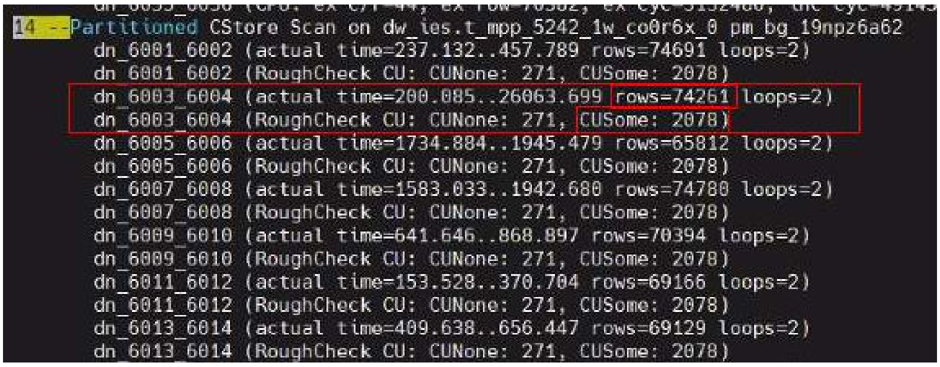
----End

6.26 Slow Performance Caused by Too Many Small CUs in Column Storage

In actual service scenarios, a large number of column-store tables are used. However, improper use of column-store tables may cause serious performance problems. The most common problem is slow performance caused by too many small CUs.

Symptom

1. The system I/O surges for a long time, and the query becomes slow occasionally.
2. After checking the execution plan information when the service is occasionally slow, it is found that the cstore scan is slow. The reason is that although the amount of data to be scanned is small, the number of CUs to be scanned is large.



```
14 --Partitioned CStore Scan on dw_ies.t_mpp_5242_1w_co0r6x_0 pm_bg_19npz6a62
      dn_6001_6002 (actual time=237.132..457.789 rows=74691 loops=2)
      dn_6001_6002 (RoughCheck CU: CUNone: 271, CUSome: 2078)
      dn_6003_6004 (actual time=200.085..26063.699 rows=74261 loops=2)
      dn_6003_6004 (RoughCheck CU: CUNone: 271, CUSome: 2078)
      dn_6005_6006 (actual time=1734.884..1945.479 rows=65812 loops=2)
      dn_6005_6006 (RoughCheck CU: CUNone: 271, CUSome: 2078)
      dn_6007_6008 (actual time=1583.033..1942.680 rows=74780 loops=2)
      dn_6007_6008 (RoughCheck CU: CUNone: 271, CUSome: 2078)
      dn_6009_6010 (actual time=641.646..868.897 rows=70394 loops=2)
      dn_6009_6010 (RoughCheck CU: CUNone: 271, CUSome: 2078)
      dn_6011_6012 (actual time=153.528..370.704 rows=69166 loops=2)
      dn_6011_6012 (RoughCheck CU: CUNone: 271, CUSome: 2078)
      dn_6013_6014 (actual time=409.638..656.447 rows=69129 loops=2)
      dn_6013_6014 (RoughCheck CU: CUNone: 271, CUSome: 2078)
```

As shown in the following figure, a CU can store 60,000 records, but more than 2000 CUs need to be scanned for 70,000 records. There are too many small CUs.

Troubleshooting

Check the data distribution in the table CUs. Perform the following operations on DNs:

1. Check the cudesc table corresponding to the column-store table.

   For non-partitioned tables:

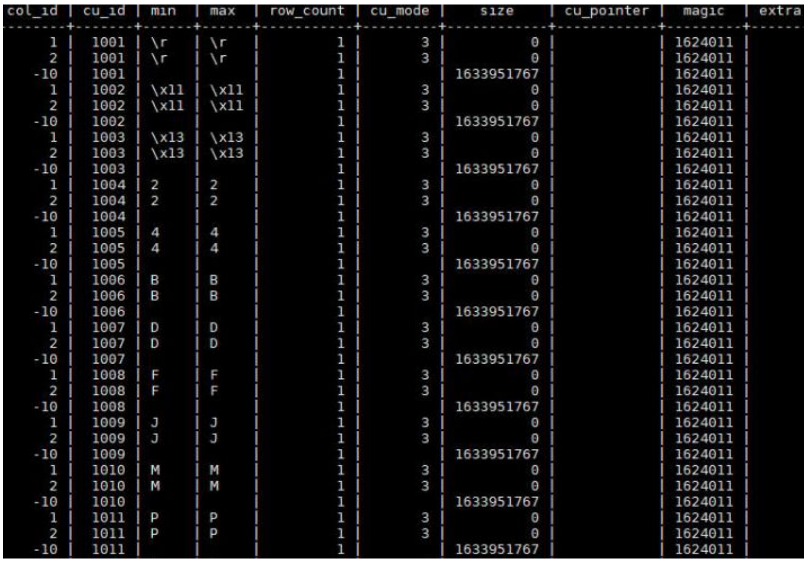
   ```
   SELECT 'cstore.'||relname FROM pg_class where oid = (SELECT relcudescrelid FROM pg_class c inner join pg_namespace n on c.relnamespace = n.oid where relname = 'table name' and nspname = 'schema name');
   ```

   For partitioned tables:

   ```
   SELECT 'cstore.'||relname FROM pg_class where oid in (SELECT p.relcudescrelid FROM pg_partition p,pg_class c,pg_namespace n where c.relnamespace = n.oid and p.parentid = c.oid and c.relname = 'table name' and n.nspname = 'schema name' and p.relcudescrelid != 0);
   ```

2. Check the rowcount of each CU in the cudesc table.

   Query the cudesc table information returned in step 1. The query result is similar to the following. Pay attention to the number of CUs whose row_count is too small (far less than 60,000). If the number is large, there are many small CUs and the CU expansion problem is serious, affecting the storage efficiency and query access efficiency.



```
col_id | cu_id | min  | max  | row_count | cu_mode |    size    | cu_pointer |  magic  | extra
-------+-------+------+------+-----------+---------+------------+------------+---------+------
     1 |  1001 | \r   | \r   |         1 |       3 |          0 |            | 1624011 |
     2 |  1001 | \r   | \r   |         1 |       3 |          0 |            | 1624011 |
   -10 |  1001 |      |      |         1 |         | 1633951767 |            | 1624011 |
     1 |  1002 | \x11 | \x11 |         1 |       3 |          0 |            | 1624011 |
     2 |  1002 | \x11 | \x11 |         1 |       3 |          0 |            | 1624011 |
   -10 |  1002 |      |      |         1 |         | 1633951767 |            | 1624011 |
     1 |  1003 | \x13 | \x13 |         1 |       3 |          0 |            | 1624011 |
     2 |  1003 | \x13 | \x13 |         1 |       3 |          0 |            | 1624011 |
   -10 |  1003 |      |      |         1 |         | 1633951767 |            | 1624011 |
     1 |  1004 | 2    | 2    |         1 |       3 |          0 |            | 1624011 |
     2 |  1004 | 2    | 2    |         1 |       3 |          0 |            | 1624011 |
   -10 |  1004 |      |      |         1 |         | 1633951767 |            | 1624011 |
     1 |  1005 | 4    | 4    |         1 |       3 |          0 |            | 1624011 |
     2 |  1005 | 4    | 4    |         1 |       3 |          0 |            | 1624011 |
   -10 |  1005 |      |      |         1 |         | 1633951767 |            | 1624011 |
     1 |  1006 | B    | B    |         1 |       3 |          0 |            | 1624011 |
     2 |  1006 | B    | B    |         1 |       3 |          0 |            | 1624011 |
   -10 |  1006 |      |      |         1 |         | 1633951767 |            | 1624011 |
     1 |  1007 | D    | D    |         1 |       3 |          0 |            | 1624011 |
     2 |  1007 | D    | D    |         1 |       3 |          0 |            | 1624011 |
   -10 |  1007 |      |      |         1 |         | 1633951767 |            | 1624011 |
     1 |  1008 | F    | F    |         1 |       3 |          0 |            | 1624011 |
     2 |  1008 | F    | F    |         1 |       3 |          0 |            | 1624011 |
   -10 |  1008 |      |      |         1 |         | 1633951767 |            | 1624011 |
     1 |  1009 | J    | J    |         1 |       3 |          0 |            | 1624011 |
     2 |  1009 | J    | J    |         1 |       3 |          0 |            | 1624011 |
   -10 |  1009 |      |      |         1 |         | 1633951767 |            | 1624011 |
     1 |  1010 | M    | M    |         1 |       3 |          0 |            | 1624011 |
     2 |  1010 | M    | M    |         1 |       3 |          0 |            | 1624011 |
   -10 |  1010 |      |      |         1 |         | 1633951767 |            | 1624011 |
     1 |  1011 | P    | P    |         1 |       3 |          0 |            | 1624011 |
     2 |  1011 | P    | P    |         1 |       3 |          0 |            | 1624011 |
   -10 |  1011 |      |      |         1 |         | 1633951767 |            | 1624011 |
```

Trigger Conditions

Column-store data is frequently imported in small batches. In scenarios where partitions are involved and the number of partitions is large, the small CU problem is more serious.

Solutions

Service Side

1. Import column-store tables in batches. The amount of data to be imported at a time (if there are partitions, the amount of data to be imported to a single partition at time) is close to or greater than 60,000 x Number of primary DNs.
2. If the data volume is small, you are advised to use row-store tables for data import.

Maintenance Portal

If the amount of data to be imported to the database cannot be adjusted on the service side, periodically perform **VACUUM FULL** on column-store tables to integrate small CUs. This will relieve the problem to some extent.

7 Cluster Exceptions

7.1 The Disk Usage Alarm Is Frequently Generated

Symptom

Alarms are generated when the disk usage of a DWS cluster reaches 80%.

Possible Causes

The alarm threshold configured for the cluster is improper.

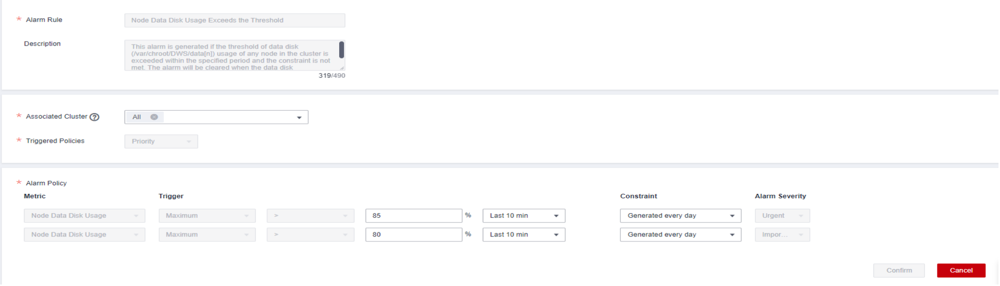
Handling Procedure

Set the triggering condition on the GaussDB (DWS) management console. You can set the disk usage, alarm duration, and frequency.

> **NOTICE**
>
> If the cluster disk usage reaches 90%, the cluster becomes read-only. You will need time to handle this issue.

1. Log in to the GaussDB(DWS) management console.
2. In the navigation pane on the left, click **Alarms**.
3. Click **View Alarm Rule** in the upper left corner.
4. On the **Alarm Rules** page that is displayed, click **Modify** in the **Operation** column of the target alarm rule. Change the trigger condition. If the average value is greater than 90%, the alarm will be triggered. Set the suppression policy to generate one alarm each day. (This example is for reference only.)
   - **Trigger**: calculation rule for threshold determination of a monitoring metric. Select the average value within a period of time of a metric to reduce the probability of alarm oscillation.
   - **Constraint**: suppresses the repeated triggering and clearance of alarms of the same type within the specified period.

Figure 7-1 Setting an alarm rule



* Alarm Rule: Node Data Disk Usage Exceeds the Threshold

Description: This alarm is generated if the threshold of data disk (/var/chroot/DWS/data[n]) usage of any node in the cluster is exceeded within the specified period and the constraint is not met. The alarm will be cleared when the data disk

319/490

* Associated Cluster: All

* Triggered Policies: Priority

* Alarm Policy

Metric	Trigger					Constraint	Alarm Severity
Node Data Disk Usage	Maximum	>	85	%	Last 10 min	Generated every day	Urgent
Node Data Disk Usage	Maximum	>	80	%	Last 10 min	Generated every day	Impor...

8 Database Use

8.1 An Error Is Reported When Data Is Inserted or Updated, Indicating that the Distribution Key Cannot Be Updated

Symptom

An error is reported when data is inserted or updated, indicating that the distribution key cannot be updated. The following is the error message:

```
ERROR: Distributed key column can't be updated in current version
```

Possible Causes

The GaussDB(DWS) distribution key cannot be updated.

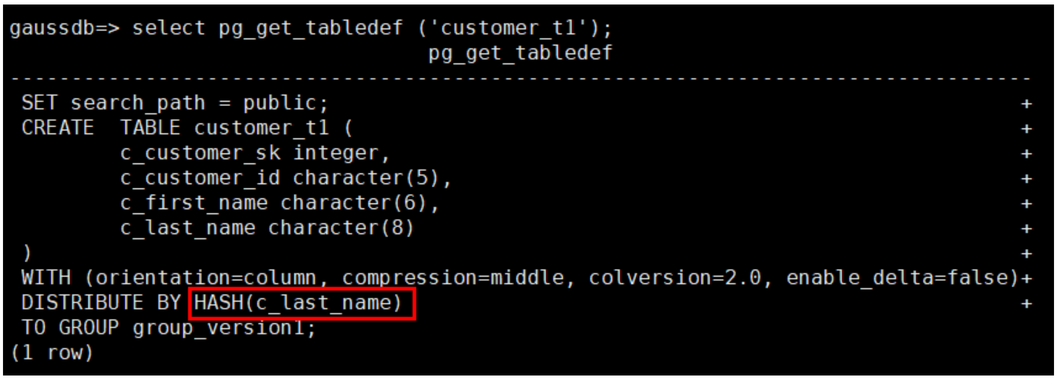
Handling Procedure

Method 1: The distribution key cannot be updated. Ignore the error.

Method 2: Change the distribution column to a column that cannot be updated. (In versions later than 8.1.0, the distribution column can be changed.) For example:

Step 1 Query the table definition. The command output shows that the distribution column of the table is **c_last_name**.

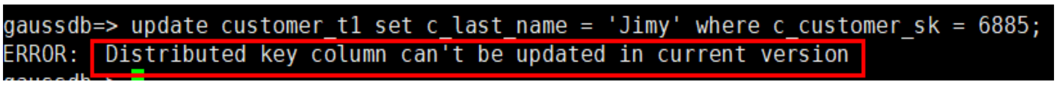
```
SELECT pg_get_tabledef('customer_t1');
```



```
gaussdb=> select pg_get_tabledef ('customer_t1');
                                  pg_get_tabledef
-------------------------------------------------------------------------------------
 SET search_path = public;                                                          +
 CREATE  TABLE customer_t1 (                                                        +
         c_customer_sk integer,                                                     +
         c_customer_id character(5),                                                +
         c_first_name character(6),                                                 +
         c_last_name character(8)                                                   +
 )                                                                                  +
 WITH (orientation=column, compression=middle, colversion=2.0, enable_delta=false)+
 DISTRIBUTE BY HASH(c_last_name)                                                    +
 TO GROUP group_version1;
(1 row)
```

Step 2 Try updating data in the distribution column. An error message will be displayed.

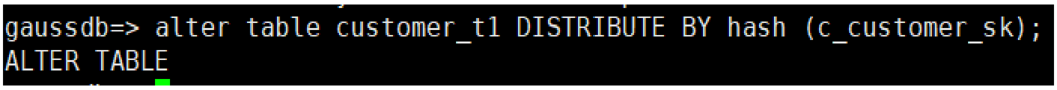
```
UPDATE customer_t1 SET c_last_name = 'Jimy' WHERE c_customer_sk = 6885;
```



```
gaussdb=> update customer_t1 set c_last_name = 'Jimy' where c_customer_sk = 6885;
ERROR:  Distributed key column can't be updated in current version
```

Step 3 Change the distribution column of the table to a column that cannot be updated, for example, **c_customer_sk**.

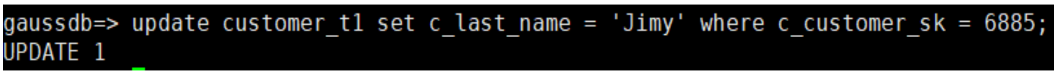
```
ALTER TABLE customer_t1 DISTRIBUTE BY hash (c_customer_sk);
```



```
gaussdb=> alter table customer_t1 DISTRIBUTE BY hash (c_customer_sk);
ALTER TABLE
```

Step 4 Update the data in the old distribution column.

```
UPDATE customer_t1 SET c_last_name = 'Jimy'WHERE c_customer_sk = 6885;
```



```
gaussdb=> update customer_t1 set c_last_name = 'Jimy' where c_customer_sk = 6885;
UPDATE 1
```

----End

8.2 "Connection reset by peer" Is Displayed When a User Executes an SQL Statement

Symptom

"Connection reset by peer" is displayed when a user executes an SQL statement.

```
ERROR:  Failed to read response from Datanodes Detail: Connection reset by peer
```

Cause Analysis

Network is disconnected due to socket communication errors under heavy network traffic.

Solution

- Set the following GUC parameters to control the peak value of the network traffic:

  comm_quota_size = 400, comm_usable_memory = 100.

To change the parameter value, perform the following steps:

a. Log in to the GaussDB(DWS) management console.

b. In the navigation tree on the left, click **Clusters**.

c. In the cluster list, find the target cluster and click the cluster name. The **Basic Information** page is displayed.

d. Go to the **Parameters** page of the cluster, find the **comm_quota_size** and **comm_usable_memory** parameters, change their values, and click **Save**. On the displayed confirmation page, check again, then click **Save**.

- After detecting such errors, the database automatically retries the SQL statements. The number of retries is controlled by **max_query_retry_times**.

NOTE

Only one SQL statement can be retried excluding error SQL statements in a transaction block.

8.3 "value too long for type character varying" Is Displayed When VARCHAR(n) Stores Chinese Characters

Symptom

The **VARCHAR(18)** field cannot store eight Chinese characters. The following error is reported:

```
org.postgresql.util.PSQLException: ERROR: value too long for type character varying(18)
```

Possible Causes

Take UTF-8 encoding as an example. A Chinese character is 3 to 4 bytes long. Eight Chinese characters are 24 to 32 bytes long, which exceeds the maximum length (18 bytes) of VARCHAR(18).

If a column contains Chinese characters, you can use the **char_length** or **length** function to query the character length and use the **lengthb** function to query the byte length.

Handling Procedure

varchar(*n*) is used to store variable length value as a string, here *n* denotes the string length in bytes. A Chinese character is usually 3 to 4 bytes long.

Increase the value length of this field based on the actual Chinese character length. For example, to store eight Chinese characters in a field, *n* must be set to at least **32**, that is, **varchar(32)**.

8.4 Case Sensitivity in SQL Statements

Symptom

The **table01** table contains the **ColumnA** field. When the SELECT statement is executed, the system displays a message indicating that the field does not exist and reports column "columna" does not exist.

```
select ColumnA from table01 limit 100;
ERROR:  column "columna" does not exist
LINE 1: select columna from TABLE_01;
               ^
CONTEXT:  referenced column: columna
```

Possible Causes

Table field names in SQL statements are case-sensitive if they are enclosed with double quotation marks. Otherwise, they are case-insensitive (are regarded as lowercase letters).

Handling Procedure

- Delete the double quotation marks from the field name if you want it case-insensitive.
- Otherwise, add the double quotation marks to the field names.

  The following is an example:

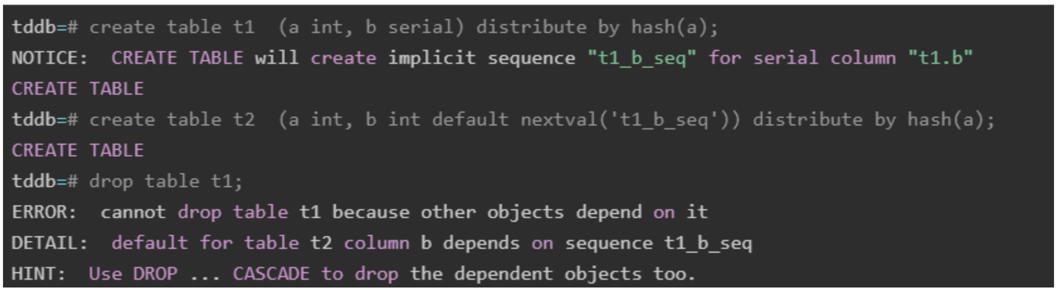
  In **table01**, when you use the SELECT statement to query **ColumnA**, add double quotation marks. The query will be successful.

  ```
  select "ColumnA" from table01 limit 100;
  ```

8.5 cannot drop table *test* because other objects depend on it Is Displayed When a Table Is Deleted

Symptom

Error **cannot drop table** *test* **because other objects depend on it** is displayed when a table is deleted, as shown in the following figure.



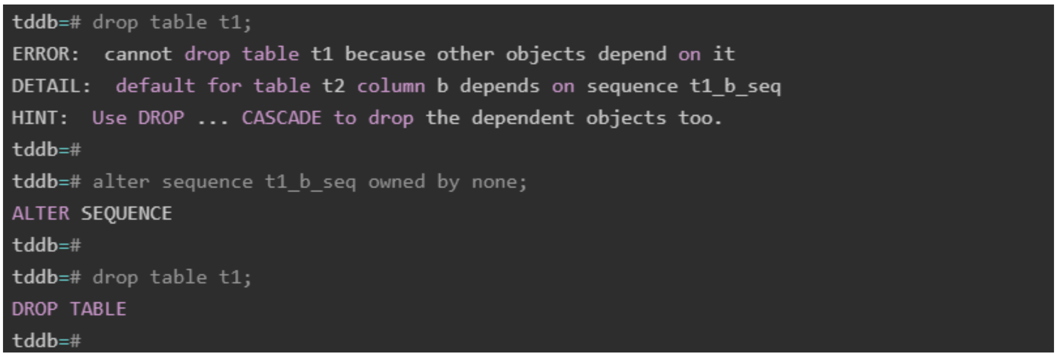
```
tddb=# create table t1  (a int, b serial) distribute by hash(a);
NOTICE:  CREATE TABLE will create implicit sequence "t1_b_seq" for serial column "t1.b"
CREATE TABLE
tddb=# create table t2  (a int, b int default nextval('t1_b_seq')) distribute by hash(a);
CREATE TABLE
tddb=# drop table t1;
ERROR:  cannot drop table t1 because other objects depend on it
DETAIL:  default for table t2 column b depends on sequence t1_b_seq
HINT:  Use DROP ... CASCADE to drop the dependent objects too.
```

Possible Causes

After table **t1** is created, a sequence is implicitly created. When table **t2** is created, the sequence is referenced. When table **t1** is deleted, the sequence will be deleted but other objects depend on the sequence. As a result, the error is reported.

Handling Procedure

If **t2** does not need to be retained, run **DROP CASCADE** to delete **t1**. If **t2** and the sequence need to be retained, delete **t1** by referring to the following figure.



```
tddb=# drop table t1;
ERROR:  cannot drop table t1 because other objects depend on it
DETAIL:  default for table t2 column b depends on sequence t1_b_seq
HINT:  Use DROP ... CASCADE to drop the dependent objects too.
tddb=#
tddb=# alter sequence t1_b_seq owned by none;
ALTER SEQUENCE
tddb=#
tddb=# drop table t1;
DROP TABLE
tddb=#
```

8.6 Failed to Execute MERGE INTO UPDATE for Multiple Tables

Symptom

Failed to execute **MERGE INTO UPDATE** for multiple tables.

Possible Causes

The following error log is printed:

```
dn_6007_6008 YY003 79375943437085786 [BACKEND] DETAIL:  blocked by hold lock thread 0, statement <pending twophase transaction>, hold lockmode (null).
```

This fault is caused by distributed locks. Two DNs lock their own data blocks and wait for the data block of the other. As a result, the locks time out.

This is a feature of two-phase locks, and this fault occurs in distributed situations.

Handling Procedure

Run the merge command to change the concurrent operations to serial operations on a single table.

8.7 JDBC Error Occurs Due to session_timeout Settings

Symptom

The following error message is displayed when using JDBC to connect to the cluster and run **COPY** to import data:

```
org.postgresql.util.PSQLException: Database connection failed when starting copy at
org.postgresql.core.v3.QueryExecutorImpl.startCopy(QueryExecutorImpl.java:804) at
org.postgresql.copy.CopyManager.copyIn(CopyManager.java:52) at
org.postgresql.copy.CopyManager.copyIn(CopyManager.java:161) at
org.postgresql.copy.CopyManager.copyIn(CopyManager.java:146) at copy.main(copy.java:95) Caused by:
java.io.EOFExceptionat org.postgresql.core.PGStream.ReceiveChar(PGStream.java:284) at
org.postgresql.core.v3.QueryExecutorImpl.processCopyResults(QueryExecutorImpl.java:1008) at
org.postgresql.core.v3.QueryExecutorImpl.startCopy(QueryExecutorImpl.java:802) ... 4 more
```

Possible Causes

session_timeout is set to **10min** by default. That is, a database is automatically disconnected if the connection is idle for more than 10 minutes.

Handling Procedure

Log in to the GaussDB(DWS) management console and set **session_timeout** to **0** or a long duration to ensure the session can stay connected for a long time.

1. Log in to the GaussDB(DWS) management console. In the cluster list, find the target cluster and click the cluster name. The **Cluster Information** page is displayed.
2. Click the **Parameter Modifications** tab and modify the value of parameter **session_timeout**. Click **Save**.

NOTE

After the COPY import is complete, you are advised to set **session_timeout** to **10 minutes**. A client connected to the database creates a thread. If the client does not perform any operation on the database for a long time, the thread is idle. If there are a large number of such clients, the connection resources will be wasted.

8.8 DROP TABLE Fails to Be Executed

Symptom

DROP TABLE fails to be executed in the following scenarios:

- A user runs the **SELECT * FROM DBA_TABLES** statement (or runs the **\dt+** command using **gsql**) and finds that the *table_name* table does not exist in the database. When the user runs the **CREATE TABLE** *table_name* statement, an error message indicating that the *table_name* table already exists. When the user runs the **DROP TABLE** *table_name* statement, an error message indicating that the *table_name* table does not exist. In this case, the *table_name* table cannot be recreated.
- A user runs the **SELECT * FROM DBA_TABLES** statement (or runs the **\dt+** command using **gsql**) and finds that the *table_name* table exists in the database. When the user runs the **DROP TABLE** *table_name* statement, an error message indicating that the *table_name* table does not exist. In this case, the *table_name* table cannot be recreated.

Cause Analysis

The table_name table exists on some nodes only.

Solution

In the preceding scenarios, if **DROP TABLE** *table_name* fails to be executed, run **DROP TABLE IF EXISTS** *table_name* to successfully drop *table_name*.

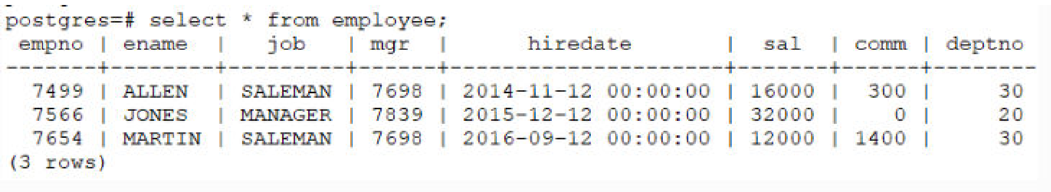
8.9 Execution Results of the string_agg Function Are Inconsistent

Symptom

The execution results of an SQL statement are inconsistent.

Possible Causes

The **string_agg** function is used in the SQL statement. The statement logic is shown in the following figure.



```
postgres=# select * from employee;
 empno | ename  |   job   | mgr  |      hiredate       |  sal  | comm | deptno
-------+--------+---------+------+---------------------+-------+------+--------
  7499 | ALLEN  | SALEMAN | 7698 | 2014-11-12 00:00:00 | 16000 |  300 |     30
  7566 | JONES  | MANAGER | 7839 | 2015-12-12 00:00:00 | 32000 |    0 |     20
  7654 | MARTIN | SALEMAN | 7698 | 2016-09-12 00:00:00 | 12000 | 1400 |     30
(3 rows)
```

Run the following SQL statement:

```
select count(*) from
(select deptno, string_agg(ename, ',') from employee group by deptno) t1 ,
(select deptno, string_agg(ename, ',') from employee group by deptno) t2
where t1.string_agg = t2.string_agg;
```

When this statement is executed repeatedly, the result sets are inconsistent (t1 or t2).

The **string_agg** function is used to concatenate data in a group into one row. However, if you use **string_agg(ename, ',')**, the order of concatenated results needs to be specified.

If the order is not specified, the output of the SQL statement above can be any one of the following:

30 | ALLEN,MARTIN

30 |MARTIN,ALLEN

Therefore, the result of subquery **t1** may be different from that of subquery **t2** when the value of **deptno** is **30**.

Handling Procedure

Add **order by** to **string_agg** to ensure that the values in the **ename** column are concatenated in the specified order.

```
select count(*) from
(select deptno, string_agg(ename, ',' order by ename desc) from employee group by deptno) t1 ,
(select deptno, string_agg(ename, ',' order by ename desc) from employee group by deptno) t2
where t1.string_agg = t2.string_agg;
```

8.10 Error "could not open relation with OID xxxx" Is Reported During Table Size Query

Symptom

When **pg_table_size** is used to query the size of a table, the error "could not open relation with OID xxxx" is reported.

Possible Causes

The table does not exist. NULL or or an error is returned.

Solution

1. Use the exception method to to ignore this error. Return **-1** for the tables that do not exist. Execute the following function:

```
CREATE OR REPLACE FUNCTION public.pg_t_size(tab_oid OID,OUT retrun_code text)
RETURNS text
LANGUAGE plpgsql
AS $$ DECLARE
v_sql text;
ts text;
BEGIN
V_SQL:='select pg_size_pretty(pg_table_size('||tab_oid||'))';
EXECUTE IMMEDIATE V_SQL into ts;
IF ts IS NULL
THEN RETRUN_CODE:=-1;
ELSE
return ts;
END IF;
EXCEPTION
WHEN OTHERS THEN
RETRUN_CODE:=-1;
END$$;
```

2. Run the following commands:

```
call public.pg_t_size('1','');
 retrun_code
-------------
 -1
(1 row)

select oid from pg_class limit 2;
 oid
------
 2662
 2659
(2 rows)

call public.pg_t_size('2662','');
 retrun_code
-------------
 120 KB
(1 row)
```

8.11 DROP TABLE IF EXISTS Syntax Errors

Symptom

The syntax of a **DROP TABLE IF EXISTS** statement is incorrect and deletes the wrong table.

Possible Causes

The operations performed by **DROP TABLE IF EXISTS** are as follows:

1. Check whether a table exists on the current CN.
2. If it does, deliver the **DROP** command to other CNs and DNs. If it does not, no operations are required.

A misunderstanding is as follows:

1. Deliver **DROP TABLE IF EXISTS** to all CNs and DNs.
2. Each CN or DN drops the table (if it exists).

Solution

If some table definitions exist in only part of the CNs/DNs, do not use the **DROP TABLE IF EXISTS** statement.

Perform the following operations instead:

1. Use the **CREATE TABLE IF NOT EXISTS** statement to complete the table definition on all the CNs and DNs.
2. If the table is no longer necessary, use the **DROP TABLE** statement on the CN to delete the table definition on all CNs and DNs.

8.12 Different Data Is Displayed for the Same Table Queried By Multiple Users

Symptom

Two users log in to the same database human_resource and run the **select count(*) from areas** statement separately to query the areas table, but obtain different results.

Cause Analysis

Check whether the two users really query the same table. In a relational database, a table is identified by three elements: **database**, **schema**, and **table**. In this issue, **database** is **human_resource** and **table** is **areas**. Then, check **schema**. Log in as users **dbadmindbadmin** and **user01** separately. It is found that **search_path** is **public** for **dbadmin** and *$user* for **user01**. By default, a schema having the same name as user dbadmin, the cluster administrator, is not created. That is, all tables will be created in **public** if no schema is specified. However, when a common user, such as **user01**, is created, the same-name schema (**user01**) is created by default. That is, all tables are created in **user01** if the schema is not specified. In conclusion, because both users performed operations on the table, there are now two different tables with the same name.

Solution

Use *schema*.**table** to determine a table for query.

8.13 When a User Specifies Only an Index Name to Modify the Index, A Message Indicating That the Index Does Not Exist Is Displayed

Symptom

Create a partitioned table index **HR_staffS_p1_index1**, without specifying index partitions.

```
CREATE INDEX HR_staffS_p1_index1 ON HR.staffS_p1 (staff_ID) LOCAL;
```

Create a partitioned table index **HR_staffS_p1_index2**, with index partitions specified.

```
CREATE INDEX HR_staffS_p1_index2 ON HR.staffS_p1 (staff_ID) LOCAL
(
    PARTITION staff_ID1_index,
    PARTITION staff_ID2_index TABLESPACE example3,
    PARTITION staff_ID3_index TABLESPACE example4
) TABLESPACE example;
```

The user changes the tablespace of an index partition **staff_ID1_index** to **example1**:

When the user executes **ALTER INDEX HR_staffS_p1_index2 MOVE PARTITION staff_ID2_index TABLESPACE example1;**, a message is displayed, indicating that the index does not exist.

Cause Analysis

Run the **CREATE INDEX HR_staffS_p1_index2 MOVE PARTITION staff_ID2_index TABLESPACE example1** command to recreate the index. An error message indicating that the index already exists is displayed. Then, run the following SQL statement or the **\d+ HR.staffS_p1** meta command using **gsql** to query the index, a message is displayed, indicating that the index already exists.

```
SELECT * FROM DBA_INDEXES WHERE index_name = HR.staffS_p1 ;
```

The possible reason why the user fails to find the index is that the user is in the public schema instead of the hr schema.

To verify this guess, execute **ALTER INDEX hr.HR_staffS_p1_index2 MOVE PARTITION staff_ID2_index TABLESPACE example1;**. The execution succeeds, proving the guess to be correct.

Execute **ALTER SESSION SET CURRENT_SCHEMA TO hr;** and then **ALTER INDEX HR_staffS_p1_index2 MOVE PARTITION staff_ID2_index TABLESPACE example1;**. The setting succeeds.

Solution

Use *schema***.table** to determine a table, index, or view for query.

8.14 An Error Is Reported During SQL Statement Execution, Indicating that the Schema Exists

Symptom

When the **CREATE SCHEMA** statement is executed, an error message is displayed, indicating that the schema exists.

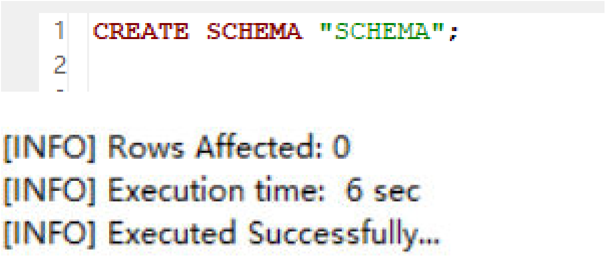
```
ERROR: schema "schema" already exists
```

Possible Causes

In SQL statements, column names are case-sensitive, and are in lowercase by default.

Handling Procedure

In case-sensitive scenarios, you need to add double quotation marks to a column name. Run the SQL statement again, as shown in the following figure. The creation is successful.



```
CREATE SCHEMA "SCHEMA";
```

```
[INFO] Rows Affected: 0
[INFO] Execution time:  6 sec
[INFO] Executed Successfully...
```

8.15 Failed to Delete a Database and an Error Is Reported Indicating that a Session Is Connected to the Database

Symptom

A database cannot be deleted and an error is reported indicating that a session is connected to the database.

Possible Causes

A session is still connected to the database, or a session keeps connecting to the database. Therefore, the database fails to be deleted. Check the database to find out whether there is a connected session. If such a session exists, find the machine that connects to the database, disconnect the connection, and delete the database.

Handling Procedure

Step 1 Using the SQL client tool to connect to the database.

Step 2 Run the following command to view the current sessions:

```
SELECT * FROM pg_stat_activity;
```

Key fields in the query result are described as follows:

- **datname**: name of the database to which the user session connects
- **usename**: name of the user who connects to the database
- **client_addr**: IP address of the client host that connects to the database

In the query result, find the name of the database to be deleted and the IP address of the corresponding client host.

Step 3 Check the host and applications that connect to the database based on the IP address of the client host, and stop the connections.

```
CLEAN CONNECTION TO ALL FOR DATABASE xxx;
```

Step 4 Run the following command to delete the database again:

```
DROP DATABASE [ IF EXISTS ] database_name;
```

----End

8.16 Byte Type Is Returned After a Table Column of the Character Type Is Read in Java

Symptom

A column in a newly created database table is of the character type. However, after the column is read in Java, the returned type is byte.

For example, create a table using the following statement:

```
CREATE TABLE IF NOT EXISTS table01(
   msg_id character(36),
   msg character varying(50)
);
```

In Java, the code for reading the field of the character type is as follows:

```
ColumnMetaInfo(msg_id,1,Byte,true,false,1,true);
```

Possible Causes

CHARACTER(n) is a fixed-length character string. When the actual string length is insufficient, the database pads it with spaces. Then, Java uses the byte type to receive the string. **CHARACTER VARYING(n)** is a variable-length character string. Java uses the string type to receive it.

8.17 "ERROR:start value of partition 'XX' NOT EQUAL up-boundary of last partition." Is Displayed When Operations Related to Table Partitions Are Performed

Symptom

When **ALTER TABLE PARTITION** is performed, the following error message is displayed:

```
ERROR:start value of partition "XX" NOT EQUAL up-boundary of last partition.
```

Cause Analysis

If the **ALTER TABLE PARTITION** statement involves both the DROP PARTITION operation and the ADD PARTITION operation, GaussDB(DWS) always performs the DROP PARTITION operation before the ADD PARTITION operation regardless of their orders. However, performing DROP PARTITION before ADD PARTITION causes a partition gap. As a result, an error is reported.

Solution

To prevent partition gaps, set **END** in DROP PARTITION to the value of **START** in ADD PARTITION.

Example: Create the partitioned table **partitiontest**.

```
CREATE TABLE partitiontest
(
 c_int integer,
 c_time TIMESTAMP WITHOUT TIME ZONE
)
PARTITION BY range (c_int)
(
 partition p1 start(100)end(108),
 partition p2 start(108)end(120)
);
```

An error is reported when the following statements are used:

```
ALTER TABLE partitiontest ADD PARTITION p3 start(120)end(130), DROP PARTITION p2;
ERROR:  start value of partition "p3" NOT EQUAL up-boundary of last partition.
ALTER TABLE partitiontest DROP PARTITION p2,ADD PARTITION p3 start(120)end(130) ;
ERROR:  start value of partition "p3" NOT EQUAL up-boundary of last partition.
```

Change them as follows:

```
ALTER TABLE partitiontest ADD PARTITION p3 start(108)end(130), DROP PARTITION p2;
ALTER TABLE partitiontest DROP PARTITION p2,ADD PARTITION p3 start(108)end(130) ;
```

8.18 Reindexing Fails

Symptom

When an index of the Desc table is damaged, a series of operations cannot be performed. The error information may be as follows:

```
index \"%s\" contains corrupted page at block
%u" ,RelationGetRelationName(rel),BufferGetBlockNumber(buf),
please reindex it.
```

Cause Analysis

In actual operations, indexes may break down due to software or hardware faults. For example, if disk space is insufficient or pages are damaged after indexes are split, the indexes may be damaged.

Solution

If the table is named **pg_cudesc_***xxxxx***_index** (indicating a column-store table), the index table of the Desc table has been damaged. Use the name of the Desc index table to find the OID and table of the corresponding primary table, and run the **REINDEX INTERNAL TABLE name** statement to reindex the **cudesc** table.

8.19 A View Failed to Be Queried

Symptom

When a user connects to a cluster database and then queries a view, the following error information is displayed:

```
[GAUSS-01850] : object with oid 16420 is not a partition object
```

Cause Analysis

The queried view is created on a partition of a partitioned table, therefore, querying this view requires access to the target partition. If this partition has been deleted, the view fails to be queried.

Solution

Step 1 Ensure that you are running an SQL statement on the view object and obtain the view name.

Check whether the object specified for **FROM** in the SQL statement is the view. If it is, record the view name.

Step 2 Delete the view using the obtained user name and schema.

Step 3 Run the SQL statement again. The target partition has been deleted, therefore, querying the view is unnecessary.

----End

8.20 Global SQL Query

The **pgxc_stat_activity** function and view are used to implement global SQL query.

1. Log in as the OS user **omm** to the host where a CN is deployed. Run **source ${BIGDATA_HOME}/mppdb/.mppdbgs_profile** to start environment variables.
2. Run the following command to connect to the database:

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

3. Run the following commands to create the **pgxc_stat_activity** function:

   ```
   DROP FUNCTION PUBLIC.pgxc_stat_activity() cascade;
   CREATE OR REPLACE FUNCTION PUBLIC.pgxc_stat_activity
   (
   OUT coorname text,
   OUT datname text,
   OUT usename text,
   OUT pid bigint,
   OUT application_name text,
   OUT client_addr inet,
   OUT backend_start timestamptz,
   OUT xact_start timestamptz,
   OUT query_start timestamptz,
   OUT state_change timestamptz,
   OUT waiting boolean,
   OUT enqueue text,
   OUT state text,
   OUT query_id bigint,
   OUT query text
   )
   RETURNS setof RECORD
   AS $$
   DECLARE
   row_data pg_stat_activity%rowtype;
   coor_name record;
   fet_active text;
   fetch_coor text;
   BEGIN
   --Get all the node names
   fetch_coor := 'SELECT node_name FROM pg_catalog.pgxc_node WHERE node_type=''C''';
   FOR coor_name IN EXECUTE(fetch_coor) LOOP
   coorname := coor_name.node_name;
   fet_active := 'EXECUTE DIRECT ON (' || coorname || ') ''SELECT * FROM pg_catalog.pg_stat_activity
   WHERE pid != pg_catalog.pg_backend_pid() and application_name not in (SELECT node_name FROM
   pg_catalog.pgxc_node WHERE node_type=''''C''''); ''';
   FOR row_data IN EXECUTE(fet_active) LOOP
   datname := row_data.datname;
   pid := row_data.pid;
   usename := row_data.usename;
   application_name := row_data.application_name;
   client_addr := row_data.client_addr;
   backend_start := row_data.backend_start;
   xact_start := row_data.xact_start;
   query_start := row_data.query_start;
   state_change := row_data.state_change;
   waiting := row_data.waiting;
   enqueue := row_data.enqueue;
   state := row_data.state;
   query_id := row_data.query_id;
   query := row_data.query;
   RETURN NEXT;
   END LOOP;
   END LOOP;
   return;
   ```

```
END; $$
LANGUAGE 'plpgsql';
```

4. Run the following command to create the **pgxc_stat_activity** view:
   ```
   CREATE VIEW PUBLIC.pgxc_stat_activity AS SELECT * FROM PUBLIC.pgxc_stat_activity();
   ```
5. Run the following SQL statement to query global session information:
   ```
   SELECT * FROM PUBLIC.pgxc_stat_activity order by coorname;
   ```

8.21 How Do I Check Whether a Table Has Been Updated or Deleted?

Symptom

You need to check for update and delete operations on a table in either of the following scenarios:

1. Frequent update or delete operations on a table generate a large number of disk page fragments and affect query performance. To improve performance, you need to identify which table has been updated, and then do VACUUM FULL to restore disk page fragments and swap OS memory.
2. To determine whether a table is a dimension table and whether it can be changed from a hash table to a replication table, check whether the table has been updated or deleted. If it does, it cannot be changed to a replication table.

Solution

Run the following commands to find the tables that have been updated or deleted:

```
ANALYZE tablename;
SELECT
    n.nspname , c.relname,
    pg_stat_get_tuples_deleted(x.pcrelid) as deleted,
pg_stat_get_tuples_updated(x.pcrelid) as updated
FROM pg_class c
INNER JOIN pg_namespace n ON n.oid = c.relnamespace
INNER JOIN pgxc_class x ON x.pcrelid = c.oid
WHERE c.relkind = 'r' and c.relname='tablename' ;
```

8.22 "Can't fit xid into page" Is Reported

Symptom

Scenario 1: Error **Can't fit xid into page, now xid is 34181619720, base is 29832807366, min is 3, max is 3.** is Reported when VACUUM FULL is executed.

Scenario 2: When the FUNCTION permission is assigned to a user at a site, the following error information is displayed:

Can't fit xid into page. relation "pg_proc", now xid is 34181619720, base is 29832807366, min is 3, max is 3.

Possible Causes

An old transaction exists in the system.

Handling Procedure

Handling procedure for scenario 1:

Step 1 Check whether there are old transactions.

```
SELECT * FROM pgxc_gtm_snapshot_status();
    xmin     |    xmax     |     csn      | oldestxmin   |  xcnt | running_xids
-------------+-------------+--------------+--------------+--------+---------------
 34730350588 | 34730350588 |  34730350553 | 34730350553  |    0
(1 row)
```

- If the value of **oldestxmin** in the query result is close to the xid **34181619720** in the error message and is greater than the values of base+min and base +max, old transactions do not affect the **FREEZE** operation. In this case, perform **Step 4**.
- If the value of **oldestxmin** in the query result is much smaller than base+min, there are old transactions in the system and VACUUM FREEZE does not take effect. In this case, perform **Step 2**.

Step 2 Run the following command to query information about old transactions in the cluster:

```
SELECT * FROM pgxc_running_xacts where xmin::text::bigint < $base+$min and xmin::text::bigint > 0;
```

Step 3 Query workloads in **Step 2** in the **pgxc_stat_activity** view and run the following command to stop the corresponding threads:

```
SELECT pg_terminate_backend(pid) FROM pgxc_running_xacts where xmin::text::bigint <$base+$min  and
xmin::text::bigint > 0;
```

NOTE

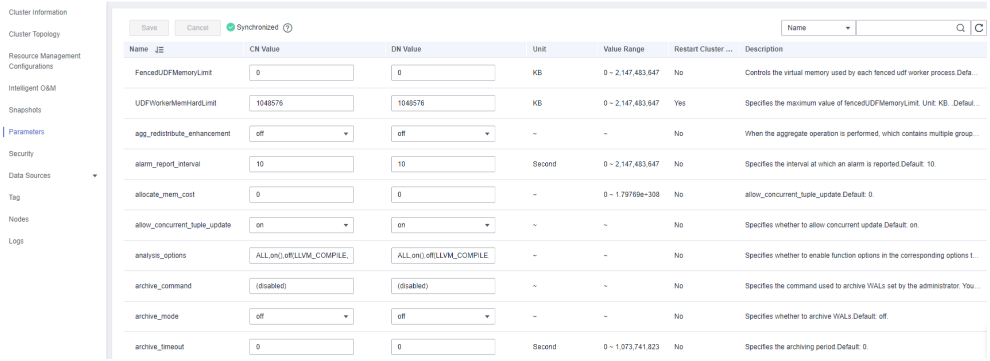
pgxc_running_xacts can only query active transactions on CNs. If the error is reported on a DN, query the **pg_running_xacts** view on the DN.

Step 4 Run the **VACUUM FULL FREEZE** statement on the table that reports the error.

```
VACUUM FULL FREEZE table_name;
```

Step 5 Log in to the GaussDB(DWS) management console and check the value of **vacuum_freeze_min_age**. If the value is **5000000000**, perform the following operations to change it to **2000000000**:

In the cluster list, find the target cluster and click the cluster name. The **Cluster Information** page is displayed. Click the **Parameters** tab and modify the value of parameter **vacuum_freeze_min_age**. Click **Save**.



Name	CN Value	DN Value	Unit	Value Range	Restart Cluster ...	Description
FencedUDFMemoryLimit	0	0	KB	0 ~ 2,147,483,647	No	Controls the virtual memory used by each fenced udf worker process.Defa...
UDFWorkerMemHardLimit	1048576	1048576	KB	0 ~ 2,147,483,647	Yes	Specifies the maximum value of fencedUDFMemoryLimit. Unit: KB. .Defaul...
agg_redistribute_enhancement	off	off	~	~	No	When the aggregate operation is performed, which contains multiple group...
alarm_report_interval	10	10	Second	0 ~ 2,147,483,647	No	Specifies the interval at which an alarm is reported.Default: 10.
allocate_mem_cost	0	0	~	0 ~ 1.79769e+308	No	allow_concurrent_tuple_update.Default: 0.
allow_concurrent_tuple_update	on	on	~	~	No	Specifies whether to allow concurrent update.Default: on.
analysis_options	ALL,on(),off(LLVM_COMPILE,	ALL,on(),off(LLVM_COMPILE	~	~	No	Specifies whether to enable function options in the corresponding options t...
archive_command	(disabled)	(disabled)	~	~	No	Specifies the command used to archive WALs set by the administrator. You...
archive_mode	off	off	~	~	No	Specifies whether to archive WALs.Default: off.
archive_timeout	0	0	Second	0 ~ 1,073,741,823	No	Specifies the archiving period.Default: 0.

----End

Handling procedure for scenario 2:

Clear the transactions by referring to **Step 2** and **Step 3** in scenario 1. You do not need to perform the **VACUUM FREEZE** operation.

8.23 "unable to get a stable set of rows in the source table" Is Reported

Symptom

MERGE INTO is used to update a destination table, or to insert a row of the source table to the destination table based on certain matching conditions. If multiple rows meet the conditions, GaussDB(DWS) may perform either the following operations:

1. Report the error "unable to get a stable set of rows in the source table".
2. A random row is inserted, which may not be the row you want.

Possible Causes

This error is reported if the MERGE INTO operation is performed to update or insert data to the destination table, and multiple rows meet the matching conditions.

Solution

Check the value of **behavior_compat_options**. If **behavior_compat_options** is set to the default value, an error will be reported when multiple rows are matched. If **behavior_compat_options** is set to **merge_update_multi**, no error will be reported, and one of the matched rows will be randomly inserted.

If the result of the MERGE INTO operation is not as expected, modify this parameter as needed, check whether multiple rows are matched, and modify the service logic.

8.24 DWS Metadata Inconsistency - Abnormal Partition Index

Symptom

The following error is reported when a user checks table definitions: "ERROR: The local index xxx on the partition xxx not exist"

Possible Causes

The primary table index is not damaged, but a partition index recorded in the **pg_partition** system catalog is inconsistent with that in other system catalogs. As a result, the partition index information cannot be found and an error is reported.

Reproducing the Issue

1. Create a table with two partitions, **p1** and **p2**.

   ```
   CREATE TABLE a_0317(a int) partition by range(a) (partition p1 values less than (4), partition p2
   values less than (8));
   ```

2. Create a primary table and its partition indexes.

   ```
   CREATE INDEX a_0317_index on a_0317(a) local (partition p1_index, partition p2_inde);
   ```

3. Check the partition index information.

   a. Check the primary table index information.

   ```
   SELECT oid,* FROM pg_class where relname ='a_0317_index';
     oid   |   relname    | relnamespace | reltype | reloftype | relowner | relam | relfilenode |
   reltablespace | relpages | reltuples | relallvisible | reltoastrelid | reltoastidxid | reldeltarelid | reld
   eltaidx | relcudescrelid | relcudescidx | relhasindex | relisshared | relpersistence | relkind | relnatts
   | relchecks | relhasoids | relhaspkey | relhasrules | relhastriggers | relhassubclass | relcmprs
    | relhasclusterkey | relrowmovement | parttype | relfrozenxid | relacl | reloptions | relreplident |
   relfrozenxid64
   --------+--------------+--------------+---------+-----------+----------+-------+-------------
   +---------------+----------+-----------+---------------+---------------+---------------+---------------
   +-----
   --------+----------------+--------------+-------------+-------------+----------------+---------+----------
   +-----------+------------+------------+-------------+----------------+----------------+---------
   -+------------------+----------------+----------+--------------+--------+------------+--------------
   +----------------
    241487 | a_0317_index |         2200 |       0 |         0 |    16393 |   403 |      241487 |            0
   |        0 |         0 |            0 |            0 |            0 |            0 |
          0 |            0 |           0 | f          | f          | p            | i       |        1 |        0 | f         |
   f         | f          | f             | f             | 0
    | f               | f            | p        | 0           |       |           | n          |            0
   (1 row)
   ```

   b. Check the partition index information based on the primary table index information.

   ```
   SELECT * FROM pg_partition where parentid= 241487;
    relname  | parttype | parentid | rangenum | intervalnum | partstrategy | relfilenode |
   reltablespace | relpages | reltuples | relallvisible | reltoastrelid | reltoastidxid | indextblid |
   indisusable |
   reldeltarelid | reldeltaidx | relcudescrelid | relcudescidx | relfrozenxid | intspnum | partkey |
   intervaltablespace | interval | boundaries | transit | reloptions | relfrozenxid64
   ----------+----------+----------+----------+-------------+--------------+-------------+---------------
   +----------+-----------+---------------+---------------+---------------+------------+-------------+-
   --------------+-------------+---------------+-------------+-------------+----------+---------
   +--------------------+----------+------------+---------+------------+----------------
    p1_index | x        |   241487 |        0 |           0 | n           |      241488 |            0 |        0 |
   0 |            0 |            0 |            0 |    241485 | t          |
   ```

```
         0 |         0 |           0 |          0 | 0          |        |        |                |
|         |       |          |           0
 p2_inde  | x       |   241487 |       0 |          0 | n          |      241489 |           0 |       0 |
0 |           0 |           0 |          0 |    241486 | t          |
         0 |         0 |           0 |          0 | 0          |        |        |                |
|         |       |          |           0
(2 rows)
```

4. Connect to a CN to start read and write transactions. Delete the index information of the **p1** partition from the **pg_partition** system catalog.

   ```
   START TRANSACTION read write;
   DELETE from pg_partition where relname = 'p1_index';
   ```

5. Check the table definition error.

   ```
   \d+ a_0317
   ERROR:  The local index 700633 on the partition 700647 not exist.CONTEXT:  referenced column:
   pg_get_indexdef
   ```

Solution

1. Delete the index information of the table.

   ```
   DROP INDEX a_0317_index;
   ```

2. Recreate the index of the table.

   ```
   CREATE INDEX a_0317_index on a_0317(a) local (partition p1_index, partition p2_inde);
   ```

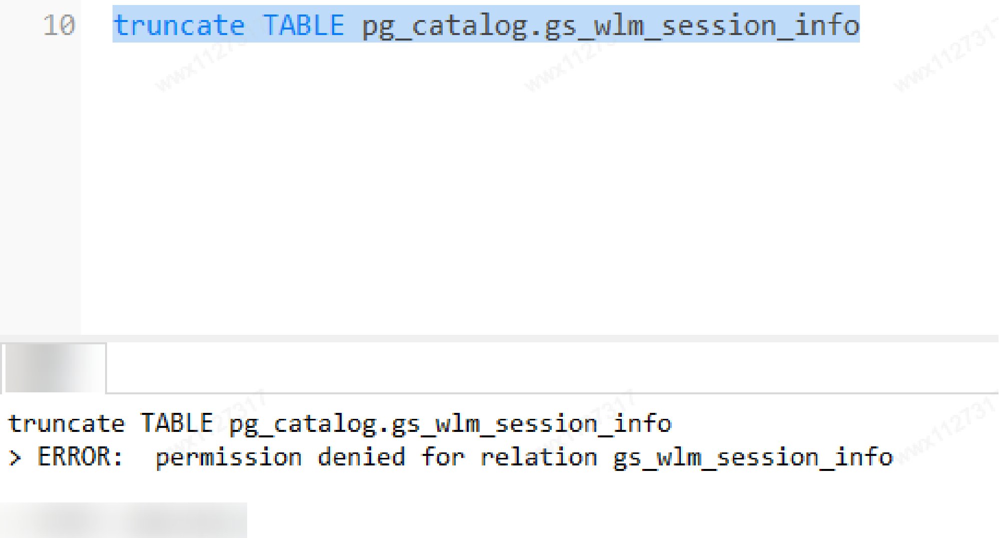
3. Check the table definition.

   ```
   \d+ a_0317
                     Table "public.a_0317"
    Column |  Type   | Modifiers | Storage | Stats target | Description
   --------+---------+-----------+---------+--------------+-------------
    a      | integer |           | plain   |              |
   Indexes:
       "a_0317_index" btree (a) LOCAL(PARTITION p1_index, PARTITION p2_inde)  TABLESPACE
   pg_default
   Range partition by(a)
   Number of partition: 2 (View pg_partition to check each partition range.)
   Has OIDs: no
   Distribute By: HASH(a)
   Location Nodes: ALL DATANODES
   Options: orientation=row, compression=no
   ```

8.25 An Error Is Reported When the truncate Command Is Executed on the System Table pg_catalog.gs_wlm_session_info

Symptom

The size of the system table **pg_catalog.gs_wlm_session_info** is large (about 20 GB). When the **truncate** command is executed on the system table without querying historical SQL statements, "permission denied for relation gs_wlm_session_info" is reported.



```
truncate TABLE pg_catalog.gs_wlm_session_info
```

```
truncate TABLE pg_catalog.gs_wlm_session_info
> ERROR:  permission denied for relation gs_wlm_session_info
```

Possible Causes

Only clusters of version 8.x or later support the **truncate** command for system tables.

To view the cluster version, log in to the GaussDB (DWS) management console, and go to the **Clusters** page. Click the name of the target cluster and view the version in the **Basic Information** page.

Handling Procedure

- To clear system tables for cluster versions earlier than 8.0, run **DELETE FROM** and then **VACUUM FULL**.

  The following uses the **gs_wlm_session_info** system table as an example:

  ```
  DELETE FROM pg_catalog.gs_wlm_session_info;
  VACUUM FULL pg_catalog.gs_wlm_session_info;
  ```

- For clusters of version 8.0 or later, run the following command to clear system tables:

  ```
  TRUNCATE TABLE dbms_om.gs_wlm_session_info;
  ```

  NOTE

  This system table schema in this example is **dbms_om**.

8.26 "inserted partition key does not map to any table partition" Is Reported When Data Is Inserted into a Partitioned Table

Symptom

"inserted partition key does not map to any table partition" is reported when data is inserted into a partitioned table.

```
CREATE TABLE startend_pt (c1 INT, c2 INT)
DISTRIBUTE BY HASH (c1)
```

```
PARTITION BY RANGE (c2) (
    PARTITION p1 START(1) END(1000) EVERY(200) ,
    PARTITION p2 END(2000),
    PARTITION p3 START(2000) END(2500) ,
    PARTITION p4 START(2500),
    PARTITION p5 START(3000) END(5000) EVERY(1000)
);
SELECT partition_name,high_value FROM dba_tab_partitions WHERE table_name='startend_pt';
 partition_name | high_value
----------------+------------
 p1_0           | 1
 p1_1           | 201
 p1_2           | 401
 p1_3           | 601
 p1_4           | 801
 p1_5           | 1000
 p2             | 2000
 p3             | 2500
 p4             | 3000
 p5_1           | 4000
 p5_2           | 5000
(11 rows)

INSERT INTO startend_pt VALUES (1,5001);
ERROR:  dn_6003_6004: inserted partition key does not map to any table partition
```

Possible Causes

In range partitioning, the table is partitioned into ranges defined by a key column or set of columns, with no overlap between the ranges of values assigned to different partitions. Data is mapped to a created partition based on the partition key value. If the data can be mapped to, it is inserted into the specific partition; if it cannot be mapped to, error messages are returned.

In this example, **partition_key** of the partitioned table tpcds.**startend_pt** is **c2**. Data inserted into the table is divided into five partitions that do not overlap. Data 5001 corresponding to column **c2** exceeds the range (5001>5000). As a result, an error is reported.

Handling Procedure

Plan partitions properly to ensure that the data can be inserted as planned.

If the planned partitions cannot meet the actual requirements, you can add partitions and then insert data. For the preceding case, you can add partition **c2**. The partition range is between **5000** and **MAXVALUE**.

```
ALTER TABLE startend_pt ADD PARTITION P6 VALUES LESS THAN (MAXVALUE);
SELECT partition_name,high_value FROM dba_tab_partitions WHERE table_name='startend_pt';
 partition_name | high_value
----------------+------------
 p1_0           | 1
 p1_1           | 201
 p1_2           | 401
 p1_3           | 601
 p1_4           | 801
 p1_5           | 1000
 p2             | 2000
 p3             | 2500
 p4             | 3000
 p5_1           | 4000
 p5_2           | 5000
 p6             | MAXVALUE
(12 rows)
```

```
INSERT INTO startend_pt VALUES (1,5001);
SELECT * FROM startend_pt;
 c1 |  c2
----+------
  1 | 5001
(1 row)
```

8.27 Error upper boundary of adding partition MUST overtop last existing partition Is Reported When a New Partition Is Added to a Range Partitioned Table

Symptom

Error **upper boundary of adding partition MUST overtop last existing partition** is reported when the **ALTER TABLE ADD PARTITION** statement is executed to add a range partition.

```
- -- Create the range partitioned table studentinfo:
CREATE TABLE studentinfo (stuno smallint, sname varchar(20), score varchar(20), examate timestamp)
PARTITION BY RANGE (examate) (
   PARTITION p1 VALUES LESS THAN ('2022-10-10 00:00:00+08'),
   PARTITION p2 VALUES LESS THAN ('2022-10-11 00:00:00+08'),
   PARTITION p3 VALUES LESS THAN ('2022-10-12 00:00:00+08'),
   PARTITION p4 VALUES LESS THAN (maxvalue)
);
- -- Add partition p0 whose boundary value is 2022-10-9 00:00:00+08:
ALTER TABLE studentinfo ADD PARTITION p0 values less than ('2022-10-9 00:00:00+08');
ERROR:  the boundary of partition "p0" is less than previous partition's boundary
- -- Add partition p5 whose boundary value is 2022-10-13 00:00:00+08:
ALTER TABLE studentinfo ADD PARTITION p5 values less than ('2022-10-13 00:00:00+08');
ERROR:  the boundary of partition "p5" is equal to previous partition's boundary
```

Possible Causes

To add a partition, the following conditions must be met:

- The name of a new partition must be different from that of an existing partition.
- The boundary value of the new partition must be greater than the upper boundary value of the last partition.
- The type of the boundary value of the new partition must be the same as that of the partition key.

The boundary of the existing partition **p1** is **(-∞, 20221010)**, and the upper boundary of the new partition **p0** is **20221009**, which falls within the partition **p1**. The boundary of the existing partition **p4** is **[20221012, +∞)**, and an upper boundary of the new partition **p5** is **20221013**, which falls within the partition **p4**. The new partitions **p0** and **p5** do not meet the conditions for running **ADD PARTITION** to add partitions. Therefore, an error is reported when the statement is executed.

Handling Procedure

You can also run the **ALTER TABLE SPLIT PARTITION** statement to split existing partitions to create more partitions. Similarly, the new partition name created by **SPLIT PARTITION** cannot be the same as that of an existing partition.

Use **SPLIT PARTITION** to split the partition **p4** with the range of **[20221012, +∞)** into partition **p4a** with the range of **[20221012, 20221013)** and partition **p4b** with the range of **[20221013, +∞)**.

```
- - Partitions before splitting.
SELECT relname, boundaries FROM pg_partition p where p.parentid='studentinfo'::regclass ORDER BY 1;
   relname   |       boundaries
-------------+-------------------------
 p1          | {"2022-10-10 00:00:00"}
 p2          | {"2022-10-11 00:00:00"}
 p3          | {"2022-10-12 00:00:00"}
 p4          | {NULL}
 studentinfo |
(5 rows)

ALTER TABLE studentinfo SPLIT PARTITION p1 AT('2022-10-09 00:00:00+08') INTO (PARTITION
P1a,PARTITION P1b);
ALTER TABLE studentinfo SPLIT PARTITION p4 AT('2022-10-13 00:00:00+08') INTO (PARTITION
P4a,PARTITION P4b);

- -- Partitions after splitting.
SELECT relname, boundaries FROM pg_partition p where p.parentid='studentinfo'::regclass ORDER BY 1;
   relname   |       boundaries
-------------+-------------------------
 p1a         | {"2022-10-09 00:00:00"}
 p1b         | {"2022-10-10 00:00:00"}
 p2          | {"2022-10-11 00:00:00"}
 p3          | {"2022-10-12 00:00:00"}
 p4a         | {"2022-10-13 00:00:00"}
 p4b         | {NULL}
 studentinfo |
(7 rows)
```

If there are requirements on the partition name, you can run the **rename partition** command to rename all partitions after splitting.

```
ALTER TABLE studentinfo RENAME PARTITION p1a to p0;
```

8.28 Error Reported During Table Query: "missing chunk number %d for toast value %u in pg_toast_XXXX"

Symptom

The error "missing chunk number %d for toast value %u in pg_toast_XXXX" is reported during table query.

Possible Causes

The data in the associated TOAST table is damaged.

TOAST is short for The OverSized Attribute Storage Technique. It is a technique for storing large column values in multiple physical rows in GaussDB(DWS). If a table contains large column values, it will have an associated TOAST table. If the OID of the table **test** is 2619, the name of the associated toast table will be **pg_toast_2619**.

Handling Procedure

Step 1 Query the damaged table based on the OID of the TOAST table (2619 in the example).

```
SELECT 2619::regclass;
   regclass
--------------
 pg_statistic
(1 row)
```

Step 2 Perform **REINDEX** and **VACUUM ANALYZE** on the located damaged table. If **REINDEX**/**VACUUM** is displayed, the repair is complete. If an error is reported during the repair, go to **Step 3**.

```
REINDEX table pg_toast.pg_toast_2619;
REINDEX table pg_statistic;
VACUUM ANALYZE pg_statistic;
```

Step 3 Run the following command to locate the damaged data row in the table.

```
DO $$
declare
 v_rec record;
BEGIN
for v_rec in SELECT * FROM pg_statistic loop
        raise notice 'Parameter is: %', v_rec.ctid;
raise notice 'Parameter is: %', v_rec;
end loop;
END;
$$
LANGUAGE plpgsql;
NOTICE:  00000: Parameter is: (46,9)
ERROR:  XX000: missing chunk number 0 for toast value 30982 in pg_toast_2619
CONTEXT:  PL/pgSQL function inline_code_block line 7 at RAISE
```

Step 4 Run the following command to delete the damaged data row located in the **Step 3**.

```
DELETE FROM pg_statistic WHERE ctid ='(46,9)';
```

Step 5 Repeat **Step 3** and **Step 4** until all incorrect data records are deleted.

Step 6 After all damaged data rows are deleted, run **REINDEX** and **VACUUM ANALYZE**, as described in **Step 2**, to repair the table again.

----End

8.29 When Inserting Data Into a Table, An Error Is Reported: "duplicate key value violates unique constraint "%s""

Symptom

When inserting data into a table, an error is reported: "duplicate key value violates unique constraint "%s"".

```
CREATE TABLE films (
code        char(5) PRIMARY KEY,
title       varchar(40) NOT NULL,
did         integer NOT NULL,
date_prod   date,
kind        varchar(10),
len         interval hour to minute
);
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "films_pkey" for table "films"
CREATE TABLE

INSERT INTO films VALUES ('UA502', 'Bananas', 105, DEFAULT, 'Comedy', '82 minutes');
INSERT INTO films VALUES ('UA502', 'Bananas', 105, '1971-07-13', 'Comedy', '82 minutes');
ERROR:  dn_6003_6004: duplicate key value violates unique constraint "films_pkey"
DETAIL:  Key (code)=(UA502) already exists.
```

Possible Causes

The table **films** is created with primary key constraint, and the **code** column is declared as the primary key. Therefore, the code column can contain only unique non-null values. In addition, the **films_pkey** index is created.

The **code** column of the inserted table contains the value **UA502** that is already in the primary key column. As a result, an error is reported.

Handling Procedure

- Method 1: Check for data conflicts and modify the inserted data. For example, change the duplicate value **UA502** to **UA509**.

  ```
  INSERT INTO films VALUES ('UA509', 'Bananas', 105, '1971-07-13', 'Comedy', '82 minutes');
  INSERT 0 1
  ```

- Method 2: Delete the primary key constraint of the table **films**.

  ```
  ALTER TABLE films DROP CONSTRAINT films_pkey;
  ALTER TABLE
  INSERT INTO films VALUES ('UA502', 'Bananas', 105, '1971-07-13', 'Comedy', '82 minutes');
  INSERT 0 1
  ```

8.30 Error could not determine which collation to use for string hashing Reported During Service Execution

Symptom

Error **could not determine which collation to use for string hashing** is reported during SELECT query.

```
CREATE TABLE t(a text collate "C", b text collate case_insensitive);
INSERT INTO t VALUES('Hello','world');
- - Calculate the hash value of ifnull(a,b).
SELECT hashtext(ifnull(a,b)) FROM t;
ERROR:  dn_6005_6006: could not determine which collation to use for string hashing.
HINT:  Use the COLLATE clause to set the collation explicitly.
```

NOTE

The **hashtext** function is used to obtain the hash value. This section is only an example to describe how to resolve a collation conflict.

Possible Causes

Table **t** contains two columns whose collation rules are different. The sorting rule of column **a** is **C** (default sorting rule during database installation), and the sorting rule of column **b** is **case_insensitive**. In the SELECT statement, the expression **hashtext(ifnull(a,b))** has multiple collations, causing a conflict. As a result, an error is reported.

Handling Procedure

If there are multiple collations in a string expression, you can manually specify **COLLATE collation_name**.

When executing SELECT, set the collation rule of the expression **ifnull(a,b)** to **C** or **case_insensitive**.

```
SELECT hashtext(ifnull(a,b) collate "C") FROM t;
 hashtext
-----------
 820977155
(1 row)

SELECT hashtext(ifnull(a,b) collate case_insensitive) FROM t;
 hashtext
-----------
 238052143
(1 row)
```

More

In the following two scenarios, multiple collations may also occur. The error messages are different, but the solutions are the same.

- Scenario 1

  In the SELECT statement, the comparison expression **a=b** has multiple collations, causing a conflict. As a result, the error **could not determine which collation to use for string comparison** is reported.

  ```
  SELECT a=b FROM t;
  ERROR:  dn_6001_6002: could not determine which collation to use for string comparison
  HINT:  Use the COLLATE clause to set the collation explicitly.
  ```

  When executing SELECT, set the collation of the expression **a=b** to **case_insensitive**.

  ```
   SELECT a=b COLLATE case_insensitive FROM t;
   ?column?
  ----------
   f
  (1 row)
  ```

- Scenario 2

  In the SELECT statement, the expression **instr(a,b)** has multiple collations, causing a conflict. As a result, the error **could not determine which collation to use for string searching** is reported.

  ```
  SELECT instr(a,b) FROM t;
  ERROR:  dn_6005_6006: could not determine which collation to use for string searching
  HINT:  Use the COLLATE clause to set the collation explicitly.
  ```

  When executing SELECT, set the collation rule of column **a** to **case_insensitive** or set the collation rule of column **b** to **C** to ensure unified collation rules.

  ```
  SELECT instr(a collate case_insensitive,b) FROM t;
   instr
  ```

```
-------
     0
(1 row)

SELECT instr(a,b collate "C") FROM t;
 instr
-------
     0
(1 row)
```

8.31 When the ODBC Driver of GaussDB(DWS) Is Used, Content of Fields of the Character Type in the SQL Query Result Is Truncated

Symptom

When the ODBC driver of GaussDB(DWS) is used, the content of fields of the character type in the SQL query result is truncated. To obtain the complete field information, use the SQL syntax **CAST BYTEA** to convert the field content into binary. However, when the same program connects to the Oracle database and SQL server, this fault does not occur.

Possible Causes

The **max varchar** parameter is set to **255** on the ODBC client. As a result, fields that have more than 255 characters are truncated.

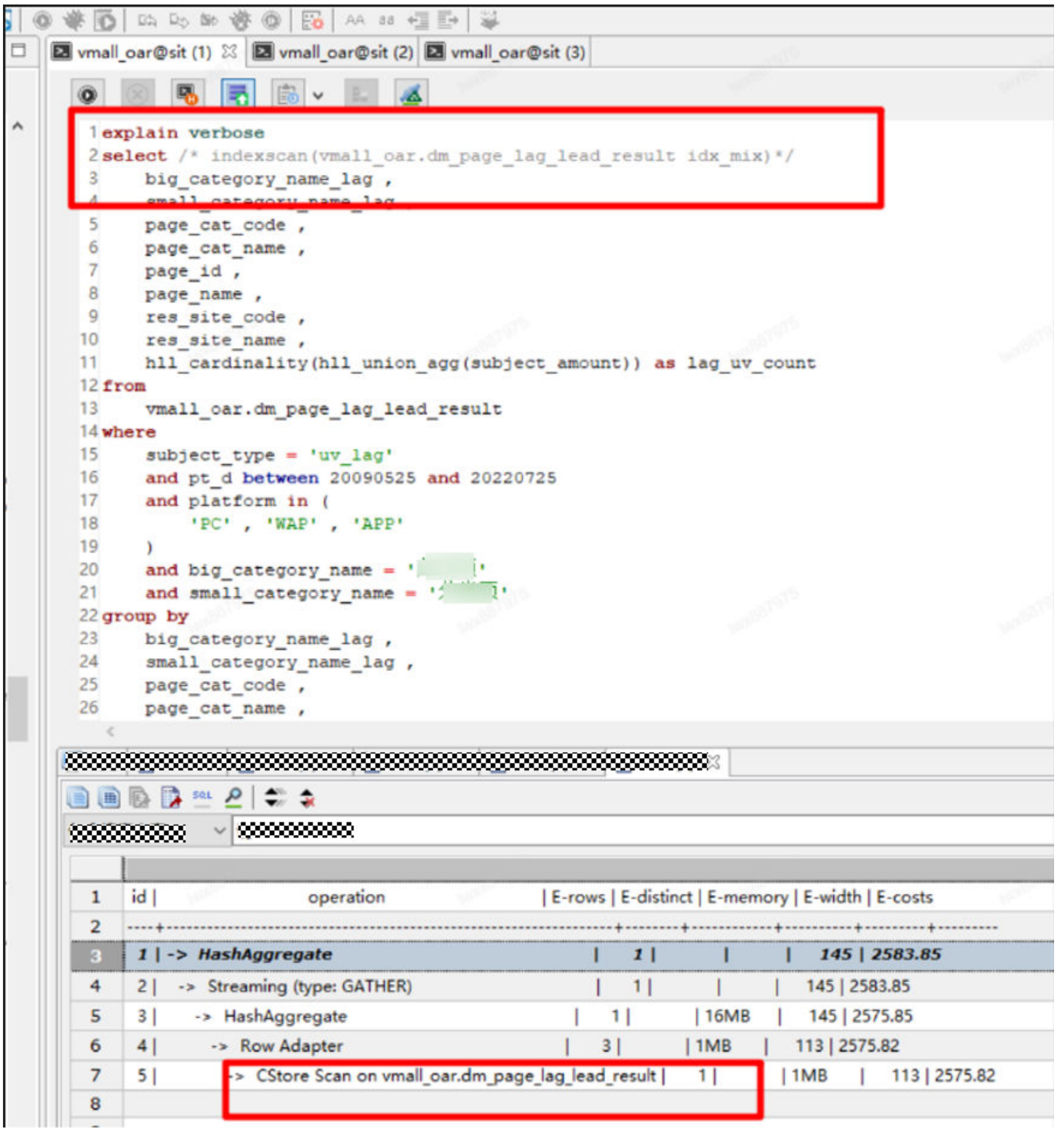
Handling Procedure

On the ODBC client, increase the value of **max varchar**.

8.32 Execution Plan Scan Hints Do Not Take Effect

Symptom

The execution plan scan hints are specified in GaussDB(DWS) but do not take effect.



```
explain verbose
select /* indexscan(vmall_oar.dm_page_lag_lead_result idx_mix)*/
    big_category_name_lag ,
    small_category_name_lag ,
    page_cat_code ,
    page_cat_name ,
    page_id ,
    page_name ,
    res_site_code ,
    res_site_name ,
    hll_cardinality(hll_union_agg(subject_amount)) as lag_uv_count
from
    vmall_oar.dm_page_lag_lead_result
where
    subject_type = 'uv_lag'
    and pt_d between 20090525 and 20220725
    and platform in (
        'PC' , 'WAP' , 'APP'
    )
    and big_category_name = '[redacted]'
    and small_category_name = '[redacted]'
group by
    big_category_name_lag ,
    small_category_name_lag ,
    page_cat_code ,
    page_cat_name ,
```

```
id |                  operation                          | E-rows | E-distinct | E-memory | E-width | E-costs
----+-----------------------------------------------------+--------+------------+----------+---------+---------
 1 | -> HashAggregate                                     |      1 |            |          |     145 | 2583.85
 2 |   -> Streaming (type: GATHER)                        |      1 |            |          |     145 | 2583.85
 3 |     -> HashAggregate                                 |      1 |            | 16MB     |     145 | 2575.85
 4 |       -> Row Adapter                                 |      3 |            | 1MB      |     113 | 2575.82
 5 |         -> CStore Scan on vmall_oar.dm_page_lag_lead_result |      1 |  | 1MB  |     113 | 2575.82
```

Possible Causes

The hint syntax is incorrect. The scan hint syntax should contain **/*+ indexscan(table_name index_name) */**. In the picture, the plus sign (+) is missing.

For details about the syntax of plan hints, see **Plan Hint Optimization Overview**.

Handling Procedure

Step 1 Add the **/*+ indexscan(table_name index_name) */** format to the SELECT statement for the scan hint syntax.

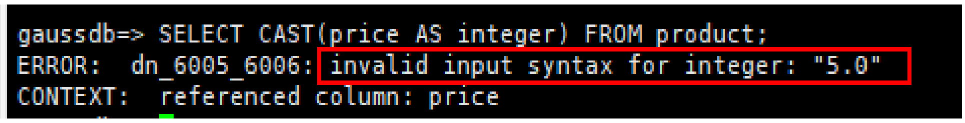
```
explain verbose
select /*+ indexscan (vmall_oar.dm_page_lag_lead_result idx_mix)*/
......
```

----End

8.33 Error "invalid input syntax for xxx" Is Reported During Data Type Conversion

Symptom

The type of a table column is varchar(20), and the data is **5.0**. When **cast(xxx as integer)** is used to convert the data to an integer, an error is reported. The error information is as follows: **invalid input syntax for integer 5.0**



```
gaussdb=> SELECT CAST(price AS integer) FROM product;
ERROR:  dn_6005_6006: invalid input syntax for integer: "5.0"
CONTEXT:  referenced column: price
```

Possible Causes

During SQL execution, if an error similar to "invalid input syntax for integer/bigint/numeric" is reported. It is most likely there is a data type conversion error. For example, the character **a** or space is converted to the integer or bigint type.

If you are familiar with the numeric and character types of GaussDB(DWS), you can avoid data type usage problems. For details, see section **Data Types**.

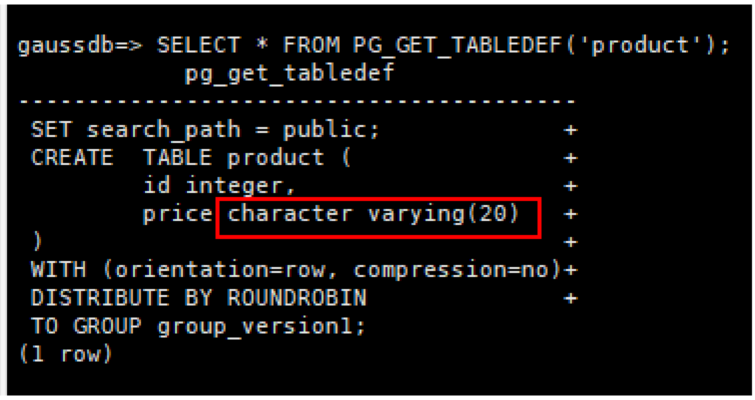
Handling Procedure

For example, if an error is reported when the string type varchar is directly converted to the integer type, you can change the column type to decimal (any precision) and then perform type conversion.

Here is the procedure:

Step 1 Assuming that the name of the error table is **product**, define the table as follows:

```
SELECT * FROM PG_GET_TABLEDEF('product');
```

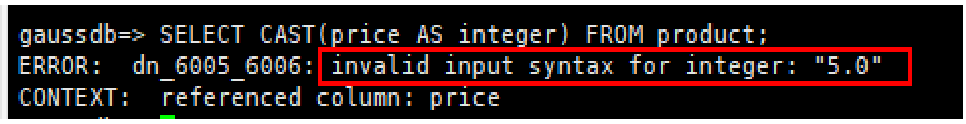


```
gaussdb=> SELECT * FROM PG_GET_TABLEDEF('product');
            pg_get_tabledef
----------------------------------------
 SET search_path = public;            +
 CREATE  TABLE product (              +
         id integer,                  +
         price character varying(20)  +
 )                                    +
 WITH (orientation=row, compression=no)+
 DISTRIBUTE BY ROUNDROBIN             +
 TO GROUP group_version1;
(1 row)
```

Step 2 Convert the query result to an integer.

```
SELECT CAST(price AS integer) FROM product;
```

The following error information is displayed:



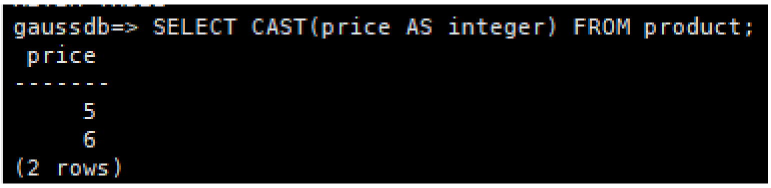
```
gaussdb=> SELECT CAST(price AS integer) FROM product;
ERROR:  dn_6005_6006: invalid input syntax for integer: "5.0"
CONTEXT:  referenced column: price
```

Step 3 Change the data type of the column to decimal.

```
ALTER TABLE product ALTER COLUMN price TYPE decimal(10,1);
```

Step 4 Data is successfully converted to an integer.

```
SELECT CAST(price AS integer) FROM product;
```



```
gaussdb=> SELECT CAST(price AS integer) FROM product;
 price
-------
     5
     6
(2 rows)
```

----End

8.34 Error UNION types %s and %s cannot be matched Is Reported

Symptom

The error **UNION types %s and %s cannot be matched** is reported when the **UNION** statement is executed.

Possible Causes

In the **UNION** branch, the formats of the output columns in the same position are different.

Handling Procedure

Fault construction

Step 1 Use the client to connect to the GaussDB(DWS) database.

Step 2 Run the following SQL statements:

```
CREATE TABLE t1(a int, b timestamp);
CREATE TABLE
CREATE TABLE t2(a int, b text);
CREATE TABLE
INSERT INTO t1 select 1, current_date;
INSERT 0 1
INSERT INTO t2 select 1, current_date;
INSERT 0 1
SELECT * FROM t1 UNION SELECT * FROM t2;
ERROR:  UNION types timestamp without time zone and text cannot be matched
LINE 1: SELECT * FROM t1 UNION SELECT * FROM t2;
                         ^
```

----End

Solution

Step 1 In the example, the types of column **b** in tables **t1** and **t2** are different. As a result, a type mismatch error is reported during the **UNION** operation. Ensure that the types of the output columns in the same position of each UNION branch match.

NOTE

Column **b** in the table **t2** is of the text type, and the inserted data is **current_date**. During the insertion, implicit type conversion is automatically performed. Therefore, no error is reported during the insertion. However, during the query, implicit conversion is not automatically performed. As a result, an error is reported.

To solve the preceding problem, ensure that the output column types of each **UNION** branch match. If they do not match, forcibly convert the output column types.

```
SELECT a,b::text FROM t1 UNION SELECT a,b FROM t2;
 a |          b
---+---------------------
 1 | 2023-02-16
 1 | 2023-02-16 00:00:00
(2 rows)
```

----End

8.35 "ERROR: Non-deterministic UPDATE" Is Reported During Update

Symptom

"ERROR: Non-deterministic UPDATE" is reported when the **UPDATE** statement is executed

```
CREATE TABLE public.t1(a int, b int) WITH(orientation = column);
CREATE TABLE

CREATE TABLE public.t2(a int, b int) WITH(orientation = column);
CREATE TABLE

INSERT INTO public.t1 VALUES (1, 1);
INSERT INTO public.t2 VALUES (1, 1),(1, 2);

UPDATE t1 SET t1.b = t2.b FROM t2 WHERE t1.a = t2.a;
ERROR: Non-deterministic
UPDATEDETAIL:  multiple updates to a row by a single query for column store table.
```

Possible Causes

If a tuple in an SQL statement is updated for multiple times, the error message "ERROR: Non-deterministic UPDATE" is displayed.

The update operation is divided into two steps:

1. Search for tuples that meet the update conditions through joins.
2. Perform the update operation.

In the preceding case, for the tuple **(1, 1)** in table **public.t1**, there are two records that meet the update condition **t1.a = t2.a** in table **public.t2**: **(1, 1)** and **(1, 2)**. According to the executor, the tuple (1, 1) in the logic table **t2** needs to be updated twice. This will result in the following two scenarios:

1. When tables **public.t1** and **public.t2** are joined, **(1, 1)** is hit first and then **(1, 2)** is hit. In this case, the tuple **(1, 1)** of **public.t1** is updated to **(1,1)** and then to **(1,2)**. The final result is **(1, 2)**.
2. When tables **public.t1** and **public.t2** are joined, **(1, 2)** is hit first and then **(1, 1)** is hit. In this case, the tuple **(1, 1)** of **public.t1** is updated to **(1,2)** and then to **(1,1)**. The final result is **(1, 1)**.

During the actual execution, the sequence of the output result set of table **public.t2** affects the final output of the **UPDATE** statement (the location of table **public.t2** in actual execution may be a complex subquery). As a result, the **execution result of the UPDATE statement is random**, which is unacceptable.

Solution

You are advised to adjust the **UPDATE** statement based on the site requirements. For example, analyze the meaning of the columns of table **public.t2** to determine the target column to be updated. In the preceding case, if you want to update the value of column **b** in **public.t1** to the maximum value in **public.t2** when the values of column **a** are the same, you can modify the logic as follows:

```
UPDATE t1 SET t1.b = t2.b_max FROM (SELECT a, max(b) AS b_max FROM t2 GROUP BY a) t2 WHERE t1.a
= t2.a;
UPDATE 1
SELECT * FROM public.t1;
 a | b
---+---
 1 | 2
(1 row)
```

8.36 Error Reported During Data Insertion: null value in column ' %s' violates not-null constraint

Symptom

The error "null value in column ' %s' violates not-null constraint" is reported when data is inserted into a table. In the message, **s %** indicates the name of the column (field) where the error occurs.

```
CREATE TABLE t1(a int, b int not null);

INSERT INTO t1 VALUES (1);
ERROR:  dn_6001_6002: null value in column "b" violates not-null constraint
```

Possible Causes

In the preceding case, if the not null constraint is set on column **b** in table **t1** when the table is created, column b cannot contain null values. If column **b** is empty when data is inserted, an error is reported.

Solutions

There are two solutions to the preceding cases:

- Solution 1: Use **ALTER TABLE** to delete the not null constraint on column **b**.

```
ALTER TABLE t1 ALTER COLUMN b DROP NOT NULL;
ALTER TABLE
```

```
INSERT INTO t1 VALUES (1);
INSERT 0 1
```

- Solution 2: Maintain the non-null (not null) constraint of column **b** but do not insert null values into column **b**.

In actual services, you can select a solution based on the site requirements.

8.37 Error "unable to get a stable set of rows in the source table"

Symptom

When **MERGE INTO** is executed to update the target table based on the matching conditions, error "unable to get a stable set of rows in the source table" is reported.

When the target table **products** is updated based on the matching condition **product_id=1502** in the source table **newproducts**, an error is reported.

```
CREATE TABLE products (product_id INTEGER,product_name VARCHAR2(60),category VARCHAR2(60));

INSERT INTO products VALUES (1501, 'vivitar 35mm', 'electrncs'),(1502, 'olympus is50', 'electrncs'),(1600, 
'play gym', 'toys');

CREATE TABLE newproducts (product_id INTEGER,product_name VARCHAR2(60),category VARCHAR2(60));

INSERT INTO newproducts VALUES (1502, 'olympus camera', 'electrncs'),(1600, 'lamaze', 'toys'),(1502, 
'skateboard', 'toy');

MERGE INTO products p
  USING newproducts np
  ON (p.product_id = np.product_id)
  WHEN MATCHED THEN
  UPDATE SET p.product_name = np.product_name, p.category = np.category WHERE np.product_id = 1502;
ERROR:  dn_6003_6004: unable to get a stable set of rows in the source tables
```

Possible Causes

In the source table **newproducts**, there are two records whose **product_id** is **1502**, and the default value of **behavior_compat_options** is used. Therefore, an error is reported when two records are matched during **MERGE INTO**.

MERGE INTO is used to update a destination table, or to insert a row of the source table to the destination table based on certain matching conditions. If multiple rows meet the conditions, GaussDB(DWS) may perform either the following operations:

1. Report the error "unable to get a stable set of rows in the source table".
2. A random row is inserted, which may not be the row you want.

Check the value of **behavior_compat_options**. If **behavior_compat_options** is set to the default value, an error will be reported when multiple rows are matched. If **behavior_compat_options** is set to **merge_update_multi**, no error will be reported, and one of the matched rows will be randomly inserted.

Therefore, if the **MERGE INTO** result is not as expected, check whether the parameter is set and whether multiple rows of data are matched. If yes, modify the service logic based on the site requirements.

Solutions

- Solution 1: Set **behavior_compat_options to merge_update_multi**.

  When multiple rows are matched in the target table, no error is reported. Instead, data in one of the matched rows are randomly used. This may cause incomplete result.

```
SET behavior_compat_options=merge_update_multi;

MERGE INTO products p
  USING newproducts np
  ON (p.product_id = np.product_id)
  WHEN MATCHED THEN
  UPDATE SET p.product_name = np.product_name, p.category = np.category WHERE np.product_id =
1502;
MERGE 1

SELECT * FROM products ;
 product_id |  product_name  | category
------------+----------------+-----------
       1501 | vivitar 35mm   | electrncs
       1502 | olympus camera | electrncs
       1600 | play gym       | toys
(3 rows)
```

- Solution 2: Modify the **MERGE INTO** matching condition.

  You are advised to select an expression with a unique result as the matching condition.

```
MERGE INTO products p
  USING newproducts np
  ON (p.product_id = np.product_id)
  WHEN MATCHED THEN
  UPDATE SET p.product_name = np.product_name, p.category = np.category WHERE np.product_id !=
1502;
MERGE 1

SELECT * FROM products;
 product_id |  product_name  | category
------------+----------------+-----------
       1501 | vivitar 35mm   | electrncs
       1502 | olympus camera | electrncs
       1600 | lamaze         | toys
(3 rows)
```

8.38 Query Results Are Inconsistent in Oracle, Teradata, and MySQL Compatibility Modes

Symptom

For a service scenario, two cluster environments run the same SQL statement on the same data volume, but get different results.

Step 1 The syntax used can be simplified as follows:

```
CREATE TABLE test (a text, b int);
INSERT INTO test values('', 1);
INSERT INTO test values(null, 1);
SELECT count(*) FROM test a, test b WHERE a.a = b.a;
```

Step 2 The execution results in the two environments are as follows:

Result 1:

```
demo_db1=> SELECT count(*) FROM test a, test b WHERE a.a = b.a;
 count
-------
     0
(1 row)
```

Result 2:

```
demo_db2=> SELECT count(*) FROM test a, test b WHERE a.a = b.a;
 count
-------
     1
(1 row)
```

----End

Possible Causes

GaussDB(DWS) supports the Oracle, Teradata, and MySQL database compatibility modes.

Null and empty strings are equal in the Oracle compatibility mode, but not equal in the TD or MySQL compatibility mode. Therefore, the preceding scenarios may be caused by different compatibility mode settings of the databases in the two environments.

You can query the **PG_DATABASE** system catalog to check the compatibility mode of the database.

```
SELECT datname, datcompatibility FROM pg_database;
```

Handling Procedure

The compatibility mode of the database is specified by the **DBCOMPATIBILITY** parameter when you create a database.

- **DBCOMPATIBILTY [=] compatibilty_type**

  Specifies the compatible database type.

- Value range: **ORA**, **TD**, and **MySQL**, indicating Oracle, Teradata, and MySQL databases, respectively.

  If this parameter is not specified during database creation, the default value **ORA** is used.

To solve the problems caused by database compatibility, you need to change the compatibility modes of the two databases to be the same. GaussDB(DWS) does not support changing the compatibility mode of an existing database using the **ALTER** statement. You can specify the compatibility mode only by creating a database.

```
CREATE DATABASE td_db DBCOMPATIBILITY ='TD';
CREATE DATABASE
```

The syntax behaviors of Oracle, Teradata, and MySQL vary according to the compatibility mode of GaussDB(DWS). For details, see **Syntax Compatibility Differences Among Oracle, Teradata, and MySQL**.